Universidade de Aveiro Departamento de Matemática
2007

Susana Paula Neves Nogueira Azevedo

Conjuntos Convexos e Algoritmos para construir Invólucros Convexos

Universidade de Aveiro Departamento de Matemática
2007

Susana Paula Neves Nogueira Azevedo

Conjuntos Convexos e Algoritmos para construir Invólucros Convexos

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Matemática, área de especialização em Ensino, realizada sob a orientação científica do Professor Doutor António Leslie Bajuelos Domínguez e da Professora Doutora Tatiana Tchemisova Cordeiro, Professores Auxiliares do Departamento de Matemática da Universidade de Aveiro.

À minha filha Joana.

o júri
presidente

Professor Doutor Domingos Moreira Cardoso
Professor Catedrático do Departamento de Matemática da Universidade de Aveiro

Professor Doutor Vladimir Alekseevitch Bushenkov
Professor Associado do Departamento de Matemática da Universidade de Évora

Professor Doutor António Leslie Bajuelos Domínguez
Professor Auxiliar do Departamento de Matemática da Universidade de Aveiro

Professora Doutora Tatiana Tchemisova Cordeiro
Professora Auxiliar do Departamento de Matemática da Universidade de Aveiro

agradecimentos

Aos meus orientadores, Professor Doutor António Leslie Bajuelos Domínguez e Professora Doutora Tatiana Tchemisova Cordeiro, pela excelente orientação, pela disponibilidade e por toda a compreensão manifestada durante a elaboração desta dissertação.

Aos meus amigos, Nuno e Tó Zé, pelos úteis esclarecimentos e pelo inesgotável apoio prestado.

palavras-chave

Conjuntos convexos, invólucros convexos, construção de invólucros convexos, algoritmo, complexidade algorítmica.

resumo

Esta dissertação tem como principal objectivo a apresentação de um conjunto de algoritmos que determinam invólucros convexos no espaço bidimensional e no espaço tridimensional. Para tal, começa-se por efectuar um estudo breve sobre conjuntos convexos, apresentando os resultados mais significativos deste tipo de conjuntos. Em seguida, são apresentados algoritmos que determinam invólucros convexos no plano, efectuando-lhes um estudo em termos de complexidade e comparando-os, sempre que oportuno, com outros algoritmos. Finalmente, é feita a extensão de dois destes algoritmos ao espaço tridimensional.

keywords

Convex sets, convex hulls, convex hulls constrution, algorithm, algorithms complexity.

abstract

The main goal of this dissertation is to present a set of algorithms that determine convex hulls in the bidimensional space and tridimensional space. To do that, is done a brief study about convex sets, presenting the most significant results about this type of sets. Next, are presented algorithms to determine convex hulls on the plane, studying their complexity and compare them with other algorithms. Finally, is made an extension of two algorithms in the tridimensional space.

Conteúdo

Lista de Símbolos e Notações

$\mathbb{N}$	conjunto dos números naturais
$\mathbb{Q}$	conjunto dos números racionais
$\mathbb{R}$	conjunto dos números reais
$\mathbb{R}^n$	espaço euclidiano de dimensão n, $n \in \mathbb{N}$
$x = (x_1, ..., x_n)$	elemento de $\mathbb{R}^n$
$\overrightarrow{xy}$	vector com extremidades x e y, $x, y \in \mathbb{R}^n$, $\overrightarrow{xy} := y - x$
$d(x, y)$	distância euclidiana entre x e y, $x, y \in \mathbb{R}^n$, $d(x, y) := \sqrt{(x_1 - y_1)^2 + ... + (x_n - y_n)^2}$
$\|x\|$	norma euclidiana de x, $x \in \mathbb{R}^n$, $\|x\| := \sqrt{x_1^2 + ... + x_n^2}$
$\langle \vec{x}, \vec{y} \rangle$	produto escalar de $\vec{x}$ por $\vec{y}$, $\vec{x}, \vec{y} \in \mathbb{R}^n$, $\langle \vec{x}, \vec{y} \rangle := x_1y_1 + ... + x_ny_n$
$\vec{x} + \vec{y}$	soma dos vectores $\vec{x}$ e $\vec{y}$, $\vec{x}, \vec{y} \in \mathbb{R}^n$, $\vec{x} + \vec{y} := (x_1 + y_1, ..., x_n + y_n)$
$\lambda\vec{x}$	produto do escalar λ, $\lambda \in \mathbb{R}$, pelo vector $\vec{x}$, $\vec{x} \in \mathbb{R}^n$, $\lambda\vec{x} := (\lambda x_1, ..., \lambda x_n)$
$int\,(A)$	interior do conjunto A, $A \subseteq \mathbb{R}^n$ (Definição 1.1.17, pág. 9)
$ir\,(A)$	interior relativo do conjunto A, $A \subseteq \mathbb{R}^n$ (Definição **??**, pág. 10)
$\overline{A}$	aderência do conjunto A, $A \subseteq \mathbb{R}^n$ (Definição 1.1.23, pág. 10)
$fr\,(A)$	fronteira do conjunto A, $A \subseteq \mathbb{R}^n$ (Definição 1.1.26, pág. 10)
A^c	complementar do conjunto A, $A \subseteq \mathbb{R}^n$, $A^c := \mathbb{R}^n \setminus A$
$dim\ A$	dimensão do conjunto A (Definição 1.1.9, pág. 7)
$\vec{a} + B$	translação do conjunto B associada ao vector $\vec{a}$, $B \in \mathbb{R}^n$, $\vec{a} \in \mathbb{R}^n$, $\vec{a} + B := \{\vec{a} + x, x \in B\}$
$CH(S)$	invólucro convexo do conjunto S, $S \subseteq \mathbb{R}^n$ (Definição 2.1.5, pág. 31)
$ext\,(S)$	conjunto dos pontos extremos do conjunto S, $S \subseteq \mathbb{R}^n$
$P = \{p_i, ..., p_m\}$	polígono simples com m vértices $p_1, ..., p_m$, $p_i \in \mathbb{R}^n$, $i = 1, ..., m$
$\lceil a \rceil$	menor inteiro maior ou igual a a, $a \in \mathbb{R}$
$\lfloor a \rfloor$	maior inteiro menor ou igual a a, $a \in \mathbb{R}$
$\|A\|$	cardinalidade (número de elementos) do conjunto A, $A \subseteq \mathbb{R}^n$

Introdução

A Geometria Computacional é um ramo das Ciências da Computação que se dedica ao desenvolvimento e ao estudo de algoritmos eficientes que permitem a resolução de problemas geométricos. Em termos históricos, a Geometria Computacional possui raízes muito profundas remetendo-se à Geometria Euclidiana Clássica que, através dos seus axiomas, permitia fazer construções mais elaboradas, utilizando apenas instrumentos tais como a régua e o compasso.

Ao longo da história da Geometria foram vários os problemas cuja resolução exigiu um procedimento algorítmico. Destaca-se, por exemplo, o Problema de Apolonius [45], no qual eram dadas três circunferências e pedia-se a construção de uma quarta circunferência tangente às circunferências dadas. Este problema impôs uma solução algorítmica, a qual foi apresentada por Euclides.

Outro problema que despertou grande interesse foi a construção de um polígono regular com n lados. Utilizando apenas as construções geométricas de Euclides, a solução deste problema foi encontrada, ainda na antiguidade, mas apenas para $3 \leq n \leq 6$. Mais tarde, Carl Gauss [45] veio a mostrar que, usando somente as construções euclidianas, não era possível encontrar um algoritmo que construísse um heptágono regular.

No início do século XX, passou-se a poder medir a *simplicidade* dos algoritmos que usavam as construções euclidianas. Para tal, eram contabilizadas as operações primitivas usadas em cada algoritmo. Assim, um algoritmo era tanto mais simples quanto menos operações primitivas usava. O conceito de *simplicidade* foi introduzido por Emile Lemoine (1902) e serviu de base a um conceito fundamental em Geometria Computacional, o conceito de *complexidade* de um algoritmo. É através da complexidade de um

algoritmo que se consegue avaliar a sua eficiência para resolver determinado problema. A Geometria Computacional é, no entanto, uma disciplina com reconhecimento muito recente. Foi em 1975 que surgiu o primeiro texto no âmbito desta disciplina. Escrito por Shamos [43], este texto atraiu muito interesse dentro da comunidade científica. Mais tarde, Preparata e Shamos deram um grande contributo com o primeiro livro de Geometria Computacional intitulado *"Computational Geometry: An Introduction"* [40]. Foi nas décadas de 80 e 90 que a Geometria Computacional mais se desenvolveu, com o aparecimento de novas técnicas, como por exemplo a divisão e conquista, que contribuíram para o aparecimento de algoritmos cada vez mais eficientes. Desde então, têm sido muitos os trabalhos e pessoas envolvidas no estudo de algoritmos que produzem soluções para certos problemas geométricos.
Quando comparada com a geometria de Euclides, a Geometria Computacional possui a vantagem de ter como principal instrumento o computador, no qual podem ser trabalhadas grandes quantidades de dados e onde os algoritmos desempenham um papel central na resolução de problemas geométricos. O facto da Informática ser um dos suportes da Geometria Computacional em permanente evolução, onde é sistemático o aparecimento de novo *Hardware* e *Software*, faz com a Geometria Computacional tenha cada vez mais sucesso na resolução dos seus problemas.

A resolução de problemas clássicos em Geometria Computacional tais como o problema do par mais próximo, da determinação de invólucros convexos, da triangulação de pontos e polígonos, diagramas de Voronoy, etc., tem contado com algoritmos cada vez mais eficientes. Este facto, torna a Geometria Computacional numa disciplina bastante atractiva em termos académicos e com aplicações directas em diversas áreas, como por exemplo:

- ◇ *Robótica*: onde os diagramas de Voronoy dão um contributo na orientação de um *robot* quando este se desloca em determinada região, ajudando na divisão dessa mesma região em sub-regiões, que permitem uma melhor orientação do *robot* [6]. O problema da determinação de invólucros convexos é também aqui usado para evitar que um *robot* colida com os objectos que o rodeiam. É evidente, que se o

invólucro convexo de um *robot* não colidir com nenhum objecto, o *robot* também não colidirá.

- ◇ *Sistemas de Informação Geográfica (GIS - Geographic Information System)*: que nos permitem, através de um conjunto de procedimentos computacionais, usufruir de uma melhor gestão do espaço e dos fenómenos que nele vão acontecendo. Estes sistemas de informação lidam no seu dia-a-dia, em termos informáticos, com muita informação traduzida em objectos geométricos (pontos, rectas, polígonos, etc.) representativos de estradas, linhas de água, caminhos de ferro, áreas de vegetação, etc. Em caso de incêndio florestal em determinada região, podemos confrontar a área florestal com a área florestal ardida e, nesse caso, a Geometria Computacional contribui com a resolução do problema sobre intersecções de objectos geométricos.

A Geometria Computacional aplica-se, ainda, em áreas tais como: Teoria dos Grafos, Processamento de Imagens, Desenho Assistido por Computador (CAD), etc.

Nesta dissertação, iremos debruçar-nos sobre o problema da *determinação do Invólucro Convexo de um conjunto finito de pontos do plano ou do espaço*. Este é um dos mais importantes problemas em Geometria Computacional e conta, actualmente, com um elevado número de algoritmos bastante eficientes na sua resolução. São cada vez mais as áreas que recorrem à Geometria Computacional para verem solucionados alguns dos seus problemas com a construção de invólucros convexos.

A presente dissertação encontra-se dividida em quatro capítulos. No primeiro capítulo, apresenta-se um estudo sucinto sobre teoria de conjuntos em $\mathbb{R}^n$, em geral, e conjuntos convexos, em particular. Pretende-se que este primeiro capítulo seja uma introdução ao capítulo seguinte, onde são abordados os invólucros convexos e suas principais propriedades. No Capítulo 3, apresenta-se um conjunto de algoritmos que permitem a resolução do problema da construção de invólucros convexos de um conjunto finito de pontos. Este capítulo encontra-se dividido em duas partes fundamentais. Na primeira parte, podemos observar um conjunto de algoritmos que resolvem o problema do invólucro convexo no plano e, na segunda parte, apresentam-se dois algoritmos que

resolvem o problema no espaço. Finalmente, o Capítulo 4 será dedicado a algumas conclusões e considerações finais.

Capítulo 1

Conjuntos Convexos

1.1 Preliminares

Seja $\mathbb{R}^n$ ($n \in \mathbb{N}$) o espaço euclidiano, cujos elementos são pontos e onde a distância entre dois dos seus elementos, $x = (x_1, ..., x_n)$ e $y = (y_1, ..., y_n)$, é dada pela fórmula:

$$d(x, y) := \sqrt{(x_1 - y_1)^2 + ... + (x_n - y_n)^2}.$$

Definição 1.1.1 *Sejam E um conjunto não vazio e $\mathbb{K}$ um corpo. Dizemos que E é um* ***espaço vectorial*** *sobre um corpo $\mathbb{K}$ se verificar as seguintes propriedades:*

i) Em E, encontra-se definida uma operação binária que se designa por adição, tal que $(E, +)$ é um grupo comutativo;

ii) Encontra-se definida uma aplicação $\mathbb{K} \times E$ para E, designada por multiplicação escalar, que a cada par $(\lambda, x) \in \mathbb{K} \times E$ faz corresponder um elemento de E tal que:

- $(\forall \alpha \in \mathbb{K})(\forall x, y \in E) \quad \alpha(x + y) = \alpha x + \alpha y$;
- $(\forall \alpha, \beta \in \mathbb{K})(\forall x \in E) \quad (\alpha + \beta)x = \alpha x + \beta x$;
- $(\forall \alpha, \beta \in \mathbb{K})(\forall x \in E) \quad \alpha(\beta x) = (\alpha\beta)x$;
- $(\forall x \in E) \quad 1x = x$, *onde 1 é a identidade de* $\mathbb{K}$.

O conjunto $\mathbb{R}^n$ munido da adição de vectores e da multiplicação de um escalar por um vector usuais é um espaço vectorial e os seus elementos são vectores.

No decorrer deste trabalho, utilizaremos a notação $\vec{x}$ para designar um elemento do espaço vectorial $\mathbb{R}^n$. Se x for um elemento do espaço n-dimensional $\mathbb{R}^n$ iremos usar, simplesmente, x.

Definição 1.1.2 *Um conjunto de vectores V de $\mathbb{R}^n$ diz-se:*

i) ***fechado para a adição****, se $\forall \vec{x}, \vec{y} \in V, \vec{x} + \vec{y} \in V$;*

ii) ***fechado para o produto por um escalar****, se $\forall \vec{x} \in V$ e $\forall \lambda \in \mathbb{R},\ \lambda \vec{x} \in V$.*

Definição 1.1.3 *Um conjunto $V \subseteq \mathbb{R}^n$ diz-se um* ***subespaço vectorial*** *do espaço vectorial $\mathbb{R}^n$ se $V \neq \emptyset$ e se é fechado para a adição e multiplicação por um escalar.*

Definição 1.1.4 *Sejam $\vec{x}_1, ..., \vec{x}_k$ k vectores do espaço vectorial $\mathbb{R}^n$. Diz-se que o vector $\vec{v} \in \mathbb{R}^n$ é uma* ***combinação linear*** *dos vectores $\vec{x}_1, ..., \vec{x}_k$ se existirem $\lambda_1, ..., \lambda_k \in \mathbb{R}$ tais que $\vec{v} = \lambda_1 \vec{x}_1 + ... + \lambda_k \vec{x_k}$. Mais, se $\lambda_i \geq 0,\ \forall i \in \{1, ..., k\}$ e $\sum_{i=1}^{k} \lambda_i = 1$, então $\vec{v} = \lambda_1 \vec{x}_1 + ... + \lambda_k \vec{x}_k$ diz-se uma* ***combinação linear convexa*** *dos vectores $\vec{x}_1, ..., \vec{x}_k$.*

Definição 1.1.5 *Chama-se* ***espaço gerado*** *por um conjunto de vectores $V = \{\vec{v}_1, ..., \vec{v}_k\}$ e representa-se por $\langle V \rangle$ ao conjunto de todas as combinações lineares de vectores de V.*

Definição 1.1.6 *Sejam $\vec{x}_1, ..., \vec{x}_k$ k vectores de $\mathbb{R}^n$. Diz-se que os* ***vectores*** *$\vec{x}_1, ..., \vec{x}_k$ são* ***linearmente independentes*** *se a única combinação linear nula de $\vec{x}_1, ..., \vec{x}_k$ é trivial, isto é, $\lambda_1 \vec{x}_1 + ... + \lambda_k \vec{x}_k = 0 \Rightarrow \lambda_1 = ... = \lambda_k = 0$.*

Se os ***vectores*** *$\vec{x}_1, ..., \vec{x}_k$, em $\mathbb{R}^n$, não forem linearmente independentes dizem-se* ***linearmente dependentes*** *e, nesse caso, o vector nulo pode ser escrito como combinação linear de $\vec{x}_1, ..., \vec{x}_k$, ou seja: $\exists\, \lambda_1, ..., \lambda_k \in \mathbb{R}$, nem todos nulos (i.e. $|\lambda_1| + ... + |\lambda_k| > 0$), tais que $\lambda_1 \vec{x}_1 + ... + \lambda_k \vec{x}_k = 0$.*

Associada à definição de vectores linearmente independentes (ou dependentes) encontra-se a definição de pontos independentes (ou dependentes).

Definição 1.1.7 *Sejam $x_1, ..., x_k$ k pontos, em $\mathbb{R}^n$. Diz-se que os* ***pontos*** *$x_1, ..., x_k$ são* ***independentes*** *(independentes afim) se $\lambda_1 x_1 + ... + \lambda_k x_k = 0$, com $\lambda_1 + ... + \lambda_k = 0 \Rightarrow \lambda_1 = ... = \lambda_k = 0$. Se os pontos não forem independentes dizem-se* ***dependentes*** *(dependentes afim).*

Note-se que para mostrar a independência dos pontos $x_1, ..., x_k$ basta-nos recorrer à Definição 1.1.6 e mostrar que os vectores $\overrightarrow{x_1 x_i} := x_i - x_1$, $i = 2, ..., k$, são linearmente independentes.

Definição 1.1.8 *Sejam $S \neq \emptyset$ um subespaço vectorial de $\mathbb{R}^n$ e $V = \{\vec{v}_1, ..., \vec{v}_k\}$, $k \geq 1$, um subconjunto de vectores de S. Diz-se que V é uma* ***base*** *de S se:*

i) $\{\vec{v}_1, ..., \vec{v}_k\}$ *são vectores linearmente independentes;*

ii) $\langle V \rangle = S$.

Definição 1.1.9 *Seja S um subespaço vectorial de $\mathbb{R}^n$, chama-se* ***dimensão*** *e designa-se por $\dim S$, à cardinalidade de uma base de S.*

É fácil provar que a dimensão de um subespaço vectorial de $\mathbb{R}^n$ não depende da escolha da base.

Definição 1.1.10 *Sejam x, y dois pontos de $\mathbb{R}^n$. Ao conjunto $[x, y] = \{\lambda x + (1 - \lambda)y, 0 \leq \lambda \leq 1\}$ chama-se* ***segmento de recta fechado*** *de extremidades x e y e ao conjunto $(x, y) = \{\lambda x + (1 - \lambda)y, 0 < \lambda < 1\}$* ***segmento de recta aberto*** *de extremidades x e y. Podemos ainda distinguir outros tipos de segmentos: o* ***segmento semi-aberto à esquerda****: $(x, y] = \{\lambda x + (1 - \lambda)y, 0 \leq \lambda < 1\}$ e o* ***segmento semi-aberto à direita****: $[x, y) = \{\lambda x + (1 - \lambda)y, 0 < \lambda \leq 1\}$.*

Definição 1.1.11 *Sejam $a \in \mathbb{R}^n$ e $r > 0$. A* ***bola aberta*** $B(a;r)$ *e a* ***bola fechada*** $B[a;r]$*, centradas em a e de raio r definem-se, respectivamente, da seguinte forma:*

i) $B(a;r) = \{x \in \mathbb{R} : \|x - a\| < r\}$*;*

ii) $B[a;r] = \{x \in \mathbb{R} : \|x - a\| \leq r\}$*, onde $\|.\|$ representa a norma euclidiana, em $\mathbb{R}^n$.*

A norma euclidiana traduz também o comprimento do vector $\overrightarrow{xy} := y - x$, com $x, y \in \mathbb{R}^n$: $\|\overrightarrow{xy}\| = \|y - x\|$.

Definição 1.1.12 *Um conjunto $A \subseteq \mathbb{R}^n$ diz-se* ***limitado*** *se existir uma bola $B \subseteq \mathbb{R}^n$, aberta ou fechada, tal que $A \subseteq B$.*

Definição 1.1.13 *Seja $A \subseteq \mathbb{R}^n$. Chama-se* ***complementar*** *de A e denota-se por A^c ao conjunto $\mathbb{R}^n \backslash A$.*

Definição 1.1.14 *Seja $A \subseteq \mathbb{R}^n$. Diz-se que A é um* ***conjunto aberto*** *se para cada $a \in A$ existir uma bola aberta $B(a;r)$ contida em A, com $r > 0$.*

Em $\mathbb{R}^n$, os conjuntos abertos gozam das seguintes propriedades:

(P_1) $\emptyset$ e $\mathbb{R}^n$ são subconjuntos abertos de $\mathbb{R}^n$;

(P_2) Se A e B são conjuntos abertos, então $A \cap B$ também é;

(P_3) Seja $(A_i)_{i \in I}$ uma família[1] de conjuntos abertos, então $\bigcup_{i \in I} A_i$ é um conjunto aberto.

Note-se que a propriedade (P_2) não pode ser generalizada para o caso de um número qualquer de conjuntos, uma vez que existem famílias infinitas de conjuntos abertos cuja intersecção não é um conjunto aberto. Considere-se, por exemplo, a família infinita dos intervalos $]-\frac{1}{i}, \frac{1}{i}[_{i \in \mathbb{N}}$. É evidente que $\bigcap_{i \in \mathbb{N}}]-\frac{1}{i}, \frac{1}{i}[= \{0\}$, que não é um aberto em $\mathbb{R}$.

[1] Sejam A_i, $i \in I$, $A_i \subseteq \mathbb{R}^n$, com $I \subseteq \mathbb{R}$. Ao conjunto de todos os conjuntos A_i chama-se família $(A_i)_{i \in I}$. Note-se que uma família de conjuntos pode ser finita ($|I| < \infty$) ou infinita ($|I| = \infty$).

Definição 1.1.15 *Seja $A \subseteq \mathbb{R}^n$. Diz-se que A é um* ***conjunto fechado*** *se o seu complementar, A^c, for um conjunto aberto.*

Observação 1.1.16 *Num conjunto fechado S cada sequência convergente tem limite em S* (**Teorema de Bolzano-Weierstrass**).

Partindo das definições de conjunto aberto e conjunto fechado podemos verificar que existem apenas dois conjuntos que são, simultaneamente, abertos e fechados: o conjunto $\emptyset$ e o próprio $\mathbb{R}^n$.

Definição 1.1.17 *Seja $A \subseteq \mathbb{R}^n$. Diz-se que $a \in A$ é* ***ponto interior*** *de A se existir algum $r > 0$ tal que $B(a, r) \subseteq A$. Ao conjunto de todos os pontos interiores do conjunto A chama-se* ***interior*** *de A e denota-se por $int\,(A)$.*

As proposições seguintes são evidentes.

Proposições 1.1.18 *Seja $A \subseteq \mathbb{R}^n$. Então:*

i) o interior de A é o maior (por inclusão) conjunto aberto nele contido;

ii) A é aberto se e só se $A = int\,(A)$.

Exemplos 1.1.19 *Os exemplos seguintes mostram como podemos aplicar a* Proposição 1.1.18 *ii) para verificar se um conjunto é aberto.*

1. *Considere-se, em $\mathbb{R}$, o conjunto $B =]0, 1]$. O interior deste conjunto é o intervalo aberto $]0, 1[$. Assim, $int\,(B) \neq B$, podendo-se concluir que B não é um conjunto aberto. Note-se ainda que B não é um conjunto fechado. Realmente, $B^c =]-\infty, 0] \cup]1, +\infty[$, logo B^c não é um conjunto aberto pois $int(B^c) = =]-\infty, 0[\cup]1, +\infty[\neq B^c$.*

2. *Considere-se, em* $\mathbb{R}^2$, *o conjunto* $C = \{(x, y) \in \mathbb{R}^2 : x^2 + y^2 \leq 3\}$. *Neste caso,* $int\,(C) = \{(x, y) \in \mathbb{R}^2 : x^2 + y^2 < 3\}$. *Logo,* $int\,(C) \neq C$, *donde se conclui que* C *não é um conjunto aberto.*

Definição 1.1.20 *Chama-se* ***variedade linear*** *a toda a translação de um subespaço vectorial, isto é, sejam* E *um subespaço vectorial de* $\mathbb{R}^n$ *e* $x \in \mathbb{R}$, $x + E$ *designa-se por variedade linear.*

Definição 1.1.21 *Dado um subconjunto de pontos de* $\mathbb{R}^n$, A, *sendo* U *a variedade linear de menor dimensão que contém* A, *designa-se por* ***interior relativo*** *de* A *e denota-se por* $ir(A)$ *o conjunto* $ir(A) = \{x \in A : \exists \epsilon > 0$ *para o qual* $B(x, \epsilon) \cup U \subseteq A\}$.

Definição 1.1.22 *Seja* $A \subseteq \mathbb{R}^n$. *Diz-se que o conjunto* A *é* ***compacto*** *se* A *for um conjunto limitado e fechado.*

Definição 1.1.23 *Seja* $A \subseteq \mathbb{R}^n$. *Ao menor (por inclusão) conjunto fechado contendo* A *chama-se* ***aderência*** *(ou fecho) de* A *e denota-se por* $\overline{A}$.

Proposição 1.1.24 *Seja* $A \subseteq \mathbb{R}^n$. *Então,* A *é um conjunto fechado se e só se* $A = \overline{A}$.

Definição 1.1.25 *Seja* $A \subseteq \mathbb{R}^n$. *Diz-se que* A *é um* ***conjunto denso****, em* $\mathbb{R}^n$, *se* $\overline{A} = \mathbb{R}^n$.

Definição 1.1.26 *Um ponto* $a \in A$ *é* ***ponto fronteira*** *de* A *se* $a \in \overline{A} \cap A^c$. *Ao conjunto dos pontos fronteira de* A *chama-se* ***fronteira de*** A *e denota-se por* $fr\,(A)$.

As proposições seguintes são evidentes.

Proposições 1.1.27 *Para cada conjunto* $A \subseteq \mathbb{R}^n$, *são válidas as seguintes afirmações.*

i) A fronteira de A é um conjunto fechado;

ii) A é um conjunto fechado se e só se $fr(A) \subset A$;

iii) A é um conjunto aberto se e só se $fr(A) \subset A^c$.

1.2 Considerações gerais sobre conjuntos convexos

Definição 1.2.1 *Um subconjunto S de $\mathbb{R}^n$ diz-se **convexo**, se dados dois quaisquer pontos x e y de S, o segmento [x,y] está contido em S: $[x, y] \subseteq S$, ou seja, para todo o $\lambda \in \mathbb{R} : 0 \leq \lambda \leq 1$ a condição $\lambda x + (1 - \lambda)y \in S$ é satisfeita.*

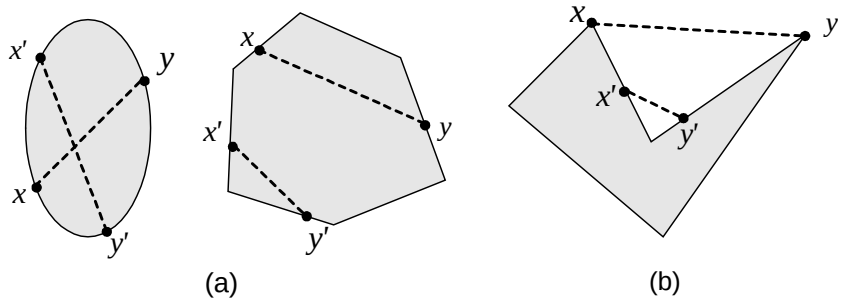


Figura 1.1: (a) Conjuntos Convexos ; (b) Conjunto não Convexo.

Exemplo 1.2.2 Como se pode observar na Figura 1.1, qualquer um dos conjuntos representados em (a) é convexo, uma vez que quaisquer que sejam dois dos seus pontos ele contém sempre o segmento de recta por eles formado. Relativamente ao conjunto representado em (b), este não é convexo pois ao considerar-se, por exemplo, os pontos x e y, o segmento de recta por eles definido não se encontra contido no conjunto.

Definição 1.2.3 *Sejam $\alpha \in \mathbb{R}$ e $\vec{a} \in \mathbb{R}^n$, com $\vec{a} \neq 0$. O conjunto $H = \{\vec{x} \in \mathbb{R}^n : \langle \vec{x}, \vec{a} \rangle = \alpha\}$ é um **hiperplano** em $\mathbb{R}^n$, onde $\langle \cdot, \cdot \rangle$ designa o produto escalar do vector $\vec{x}$ pelo vector $\vec{a}$, em $\mathbb{R}^n$.*

Definição 1.2.4 *Sejam $\vec{b} \in \mathbb{R}^n$ e $\beta \in \mathbb{R}$. O subconjunto definido por $\{\vec{x} : \langle \vec{x}, \vec{b} \rangle < \beta\}$ (ou $\{\vec{x} : \langle \vec{x}, \vec{b} \rangle > \beta\}$) designa-se por* ***semi-espaço aberto*** *de $\mathbb{R}^n$, e o subconjunto $\{\vec{x} : \langle \vec{x}, \vec{b} \rangle \leq \beta\}$ (ou $\{\vec{x} : \langle \vec{x}, \vec{b} \rangle \geq \beta\}$) por* ***semi-espaço fechado*** *de $\mathbb{R}^n$.*

Considerando a Definição 1.2.3, é fácil verificar que cada hiperplano $\{\vec{x} \in \mathbb{R}^n : \langle \vec{x}, \vec{a} \rangle = \alpha\}$ determina, em $\mathbb{R}^n$, dois semi-espaços que podem ser abertos ou fechados, por exemplo, $\{\vec{x} \in \mathbb{R}^n : \langle \vec{x}, \vec{a} \rangle \leq \alpha\}$ e $\{\vec{x} \in \mathbb{R}^n : \langle \vec{x}, \vec{a} \rangle \geq \alpha\}$, ou $\{\vec{x} \in \mathbb{R}^n : \langle \vec{x}, \vec{a} \rangle < \alpha\}$ e $\{\vec{x} \in \mathbb{R}^n : \langle \vec{x}, \vec{a} \rangle > \alpha\}$, sendo este hiperplano a fronteira dos semi-espaços em causa.

O complementar H^c de um hiperplano H pode ser visto como a união de dois semi--espaços convexos, abertos e disjuntos: $H^c = \{\vec{x} \in \mathbb{R}^n : \langle \vec{x}, \vec{a} \rangle < \alpha\} \cup \{\vec{x} \in \mathbb{R}^n : \langle \vec{x}, \vec{a} \rangle > > \alpha\}$.

Um hiperplano assume várias representações dependendo da dimensão do espaço onde este se encontra. Por exemplo, em $\mathbb{R}^2$, um hiperplano representa-se recorrendo a uma equação da forma $a_1 x + a_2 y = \alpha$, que é uma recta e, nesse caso, irá dividir $\mathbb{R}^2$ em dois semi-planos.

As bolas e os semi-espaços constituem exemplos importantes de conjuntos convexos, em $\mathbb{R}^n$. É simples confirmar a convexidade destes conjuntos.

Exemplo 1.2.5 *A bola fechada B[a;r], em $\mathbb{R}^n$, é um conjunto convexo.*

De facto, sejam $x, y \in B[a; r]$ e $0 \leq \lambda \leq 1$. Pela Definição 1.1.11 *ii*) temos que $\|x - a\| \leq r$ e $\|y - a\| \leq r$. Verifiquemos se o ponto $z = \lambda x + (1 - \lambda)y$ pertence à bola $B[a; r]$. Tendo por base as propriedades da norma, obtemos: $\|z - a\| = \|\lambda x + (1 - \lambda)y - -a\| = \|\lambda x + y - \lambda y - a + \lambda a - \lambda a\| \leq \lambda \|x - a\| + (1 - \lambda)\|y - a\| \leq \lambda r + (1 - \lambda)r \leq r$. Consequentemente, $z \in B[a; r]$ e pela Definição 1.2.1, temos que $B[a; r]$ é um conjunto convexo.

De forma análoga mostra-se a convexidade da bola aberta $B(a; r)$.

Exemplo 1.2.6 *Seja $\vec{a} \in \mathbb{R}^n \setminus \{0\}$, $\alpha \in \mathbb{R}$. O semi-espaço fechado A de R^n definido por $A := \{\vec{x} : \langle \vec{x}, \vec{a} \rangle \leq \alpha\}$ é um conjunto convexo.*

Sejam $\vec{x}, \vec{y} \in A$ e $0 \leq \lambda \leq 1$. Pretende-se mostrar que $\lambda\vec{x} + (1-\lambda)\vec{y} \in A$. Neste caso, $\langle \vec{x}, \vec{a} \rangle \leq \alpha$ e $\langle \vec{y}, \vec{a} \rangle \leq \alpha$. Utilizando as propriedades do produto escalar em $\mathbb{R}^n$, tem-se que $\langle \lambda\vec{x} + (1-\lambda)\vec{y}, \vec{a} \rangle = \lambda\langle \vec{x}, \vec{a} \rangle + (1-\lambda)\langle \vec{y}, \vec{a} \rangle \leq \lambda\alpha + (1-\lambda)\alpha = \alpha$. Logo, $\lambda\vec{x} + (1-\lambda)\vec{y} \in A$, ou seja, A é um conjunto convexo.

A prova de que um semi-espaço aberto $\{\vec{x} : \langle \vec{x}, \vec{a} \rangle < \alpha\}$ é um conjunto convexo pode ser feita de forma semelhante.

Observação 1.2.7 *Os conjuntos $\emptyset$ e $\mathbb{R}^n$ são conjuntos trivialmente convexos, em $\mathbb{R}^n$.*

Em seguida, apresentamos algumas propriedades dos conjuntos convexos.

Teorema 1.2.8 *Em $\mathbb{R}^n$, a intersecção de uma família de conjuntos convexos é um conjunto convexo.*

Demonstração: Seja $(A_i)_{i \in I}$ uma família de conjuntos convexos, em $\mathbb{R}^n$, $I \subseteq \mathbb{R}$.

Se $x, y \in \bigcap_{i \subseteq I}(A_i)$ e $0 \leq \lambda \leq 1$, então $x, y \in A_i$, $\forall i \in I$.

Como A_i é convexo, $\lambda x + (1-\lambda)y \in A_i$, para cada $i \in I$. Assim, $\lambda x + (1-\lambda)y \in \bigcap_{i \in I}(A_i)$, o que mostra que a intersecção de conjuntos convexos continua a ser um conjunto convexo.
$\diamondsuit$

Do Teorema 1.2.8 resulta que a intersecção é uma operação que preserva a convexidade. Por outro lado, a união de conjuntos convexos nem sempre é um conjunto convexo, como ilustra o exemplo seguinte.

Exemplo 1.2.9 *Em* $\mathbb{R}^2$*, considerem-se os conjuntos:* $A = \{(x, y) : x^2 + y^2 \leq 1\}$ *e* $B = \{(x, y) : (x - 3)^2 + y^2 \leq 1\}$.

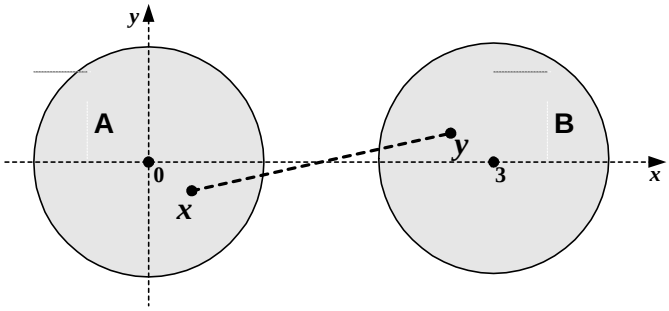


Figura 1.2: Representação geométrica dos conjuntos A e B.

É evidente que A e B são conjuntos convexos. Entretanto, a Figura 1.2 mostra que a intersecção do conjunto A com o conjunto B é o conjunto $\emptyset$, que é um conjunto trivialmente convexo. Mostra ainda que, se considerarmos a união do conjunto A com o conjunto B e se escolhermos um ponto $x \in A$ e um outro ponto $y \in B$, verificamos que o segmento de recta $[x, y]$ não se encontra contido na união de A e B. Donde se conclui que a união dos conjuntos convexos A e B não é um conjunto convexo.

Definição 1.2.10 *Um conjunto* P*, em* $\mathbb{R}^n$*, é um* ***poliedro*** *se é a intersecção de um número finito de semi-espaços fechados.*

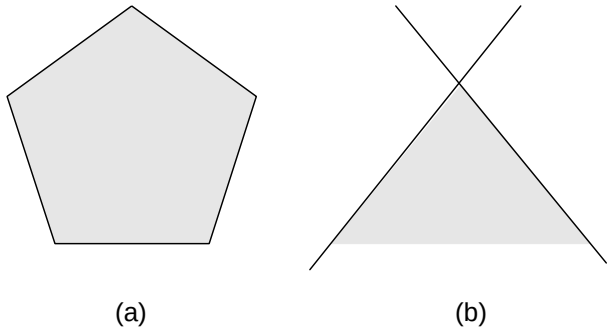


Figura 1.3: (a) Poliedro limitado ; (b) Poliedro não limitado.

Tendo em conta a definição de poliedro, podemos distinguir dois tipos de poliedros: os poliedros limitados e os poliedros não limitados, tal como se pode observar na Figura 1.3.

Os poliedros são exemplos importantes de conjuntos convexos. A verificação de que um poliedro é um conjunto convexo é muito simples e tem por base o Teorema 1.2.8. Vimos anteriormente que os semi-espaços são conjuntos convexos. Como a intersecção de conjuntos convexos é um conjunto convexo e como um poliedro é a intersecção de semi-espaços fechados (Definição 1.2.10), então um poliedro é um conjunto convexo.

O teorema seguinte é uma consequência da definição de conjunto convexo.

Teorema 1.2.11 *Sejam $x_1, ..., x_m$ elementos de um conjunto convexo S, $S \subseteq R^n$, e sejam $\lambda_1, ..., \lambda_m$ escalares não negativos tais que $\sum_{i=1}^{m} \lambda_i = 1$. Então, $\lambda_1 x_1 + ... + \lambda_m x_m \in$ $\in S$.*

Demonstração: Apliquemos a indução sobre k.
Para $k = 1$ a afirmação do teorema é evidente. Suponhamos, então, que a afirmação do teorema é válida para $k \geq 1$.
Vamos verificar que todas as combinações convexas dos $k+1$ elementos de S pertencem a S.
Seja $x = \lambda_1 x_1 + ... + \lambda_{k+1} x_{k+1}$, com $\lambda_i \geq 0$ e $\sum_{i=1}^{k+1} \lambda_i = 1$. É evidente que pelo menos um dos λ_i $(i = 1, ..., k+1)$ é diferente de um. Considere-se, por conveniência, $\lambda_1 \neq 1$. Logo, $\lambda_1 < 1$ e $1 - \lambda_1 = \lambda_2 + ... + \lambda_{k+1} > 0$.
Seja $y = \lambda_2' x_2 + ... + \lambda_{k+1}' x_{k+1}$, onde $\lambda_i' = \frac{\lambda_i}{1-\lambda_1}$, $i = 2, ..., k+1$. Então, $\lambda_i' \geq 0$, para $i = 2, ..., k+1$, e $\lambda_2' + ... + \lambda_{k+1}' = \frac{\lambda_2}{1-\lambda_1} + ... + \frac{\lambda_{k+1}}{1-\lambda_1} = \frac{\lambda_2 + ... + \lambda_{k+1}}{\lambda_2 + ... + \lambda_{k+1}} = 1$. Assim, y será combinação convexa dos k elementos de S e, por hipótese de indução, $y \in S$. Como $x = \lambda_1 x_1 + \lambda_2 x_2 + ... + \lambda_{k+1} x_{k+1} = \lambda_1 x_1 + (1-\lambda_1)[\frac{\lambda_2}{1-\lambda_1} x_2 + ... + \frac{\lambda_{k+1}}{1-\lambda_1} x_{k+1}] = \lambda_1 x_1 + (1-\lambda_1) y$ e $x_1, y \in S$, concluímos que $x \in S$. $\diamondsuit$

O Teorema 1.2.11 admite a seguinte generalização.

Teorema 1.2.12 *Um conjunto $S \subseteq \mathbb{R}^n$ é convexo se e só se S contém todas as combinações convexas de todos os seus subconjuntos finitos.*

Definição 1.2.13 *Sejam A e B conjuntos de $\mathbb{R}^n$ e $\alpha \in \mathbb{R}$. As operações adição dos conjuntos A e B e a multiplicação de um conjunto A por um escalar α definem-se, respectivamente, por:*

i) $A + B = \{x : x = x_a + x_b, x_a \in A, x_b \in B\}$;

ii) $\alpha A = \{x : x = \alpha x_a, x_a \in A, \alpha \in \mathbb{R}\}$.

As operações adição de dois conjuntos e multiplicação de um conjunto por um escalar também preservam a convexidade.

Teorema 1.2.14 *Se A e B são conjuntos convexos de $\mathbb{R}^n$ e $\alpha \in \mathbb{R}$, então:*

i) $A + B$ *é um conjunto convexo;*

ii) αA *é um conjunto convexo.*

Demonstração: (i) Se $A = \emptyset$ ou $B = \emptyset$, o resultado é imediato, uma vez que $A+\emptyset = A$, $\forall A \subseteq \mathbb{R}^n$.
Considere-se, então, $A \neq \emptyset$ e $B \neq \emptyset$. Sejam $x, y \in A + B$ e $\lambda \in [0, 1]$. Pela Definição 1.2.13, $x = x_a + x_b$, com $x_a \in A$ e $x_b \in B$, e $y = y_a + y_b$, com $y_a \in A$ e $y_b \in B$. Seja $z = (1 - \lambda)x + \lambda y$. Então,

$$\begin{aligned} z &= (1-\lambda)x + \lambda y = (1-\lambda)(x_a + x_b) + \lambda(y_a + y_b) = \\ &= (1-\lambda)x_a + \lambda y_a + (1-\lambda)x_b + \lambda y_b \end{aligned}$$

Designe-se $z_a = (1-\lambda)x_a + \lambda y_a$ e $z_b = (1-\lambda)x_b + \lambda y_b$. É evidente que $z_a \in A$ e $z_b \in B$, atendendo à convexidade dos conjuntos A e B. Logo, $z \in A + B$, com $z = z_a + z_b$. Consequentemente, $A + B$ é um conjunto convexo.

(ii) Se $A = \emptyset$, o resultado é imediato.
Considere-se, então, $A \neq \emptyset$. Sejam $x, y \in \alpha A$ e $\alpha \in [0, 1]$. Por hipótese, $x = \alpha a_1$ e

$y = \alpha a_2$, com $a_1, a_2 \in A$.

Considerando $z = (1-\lambda)x + \lambda y$ temos que,

$$z = (1-\lambda)(\alpha a_1) + \lambda(\alpha a_2) = \alpha((1-\lambda)a_1 + \lambda a_2) = \alpha a,$$

onde $a = (1-\lambda)a_1 + \lambda a_2 \in A$, atendendo à convexidade do conjunto A. Logo, $z \in \alpha A$ e, portanto, αA é um conjunto convexo. $\diamondsuit$

Corolário 1.2.15 *Sejam $X_1, ..., X_k$ conjuntos convexos, em $\mathbb{R}^n$, $k \in \mathbb{N}$. A combinação linear destes conjuntos, $\sum_{i=1}^{k} \lambda_i X_i$, $\lambda_i \in \mathbb{R}$, é um conjunto convexo.*

Teorema 1.2.16 *Seja S um conjunto convexo, em $\mathbb{R}^n$, e a um ponto arbitrário de $\mathbb{R}^n$. Então $a + S$ é um conjunto convexo.*

A demonstração deste resultado é análoga à demonstração do Teorema 1.2.14 e pode ser encontrada em [42], por exemplo.

Teorema 1.2.17 *Se $S \subseteq \mathbb{R}^n$ é um conjunto convexo, então $int\,(S)$ e $\overline{S}$ são conjuntos convexos.*

Demonstração: Sejam $x_1, x_2 \in int(S)$, $\alpha \in]0,1[$ e seja $x_3 = \alpha x_1 + (1-\alpha)x_2$. Deste modo, $x_3 \in]x_1, x_2[$.

Para provar que $int(S)$ é um conjunto convexo basta mostrar que $x_3 \in int(S)$. Escolha-se um raio $r > 0$ tal que $B(x_2, r) \subset S$. Vamos mostrar que $B(x_3, (1-\alpha)r) \subset S$, isto é, se $x \in B(x_3, (1-\alpha)r)$, então $x \in S$.

É evidente que $\frac{\|x_3 - x_1\|}{\|x_2 - x_1\|} = \frac{\|\alpha x_1 + (1-\alpha)x_2 - x_1\|}{\|x_2 - x_1\|} = \frac{\|(\alpha-1)x_1 + (1-\alpha)x_2\|}{\|x_2 - x_1\|} = |1-\alpha| = 1-\alpha$, para cada $x \in \mathbb{R}^n$, donde

$$\|x_2 - x_1\| = \frac{\|x_3 - x_1\|}{1-\alpha}. \tag{1.2.1}$$

Seja $x \in B(x_3, (1-\alpha)r)$. Neste caso, $\|x_3 - x\| < (1-\alpha)r$ e tendo em conta (1.2.1) obtemos que $\|x_2 - x\| < r$ e $x \in B(x_2, r)$, donde se conclui que $x \in int(S)$.

Agora, mostre-se que se S é convexo, então $\overline{S}$ é convexo.

Sejam $x_1, x_2 \in \overline{S}$, $x_3 = \alpha x_1 + (1-\alpha)x_2$, com $\alpha \in [0,1]$. Em S, escolham-se duas sucessões (x_{1_k}) e (x_{2_k}) convergentes, respectivamente, para x_1 e x_2. Seja $x_k = \alpha x_{1_k} + (1-\alpha)x_{2_k}$. É claro que $\lim_{k\to\infty} x_k = \lim_{k\to\infty} \alpha x_{1_k} + (1-\alpha)x_{2_k} = \alpha \lim_{k\to\infty}(x_{1_k}) + (1-\alpha)\lim_{k\to\infty}(x_{2_k}) = \alpha x_1 + +(1-\alpha)x_2 = x_3 \in \overline{S}$. Consequentemente, $x_3 = \lim_{x\to\infty} x_k$ pelo que, de acordo com a Observação 1.1.16, $x_3 = \overline{S}$. ♢

1.3 Separação de conjuntos convexos. Hiperplanos de suporte

Definição 1.3.1 *Sejam A e B dois conjuntos e H um hiperplano, em $\mathbb{R}^n$. Diz-se que H* ***separa*** *A de B se A está contido num dos semi-espaços fechados determinados por H e B no outro semi-espaço fechado. Neste caso, o hiperplano H chama-se* ***hiperplano separador*** (ver Figura 1.4).

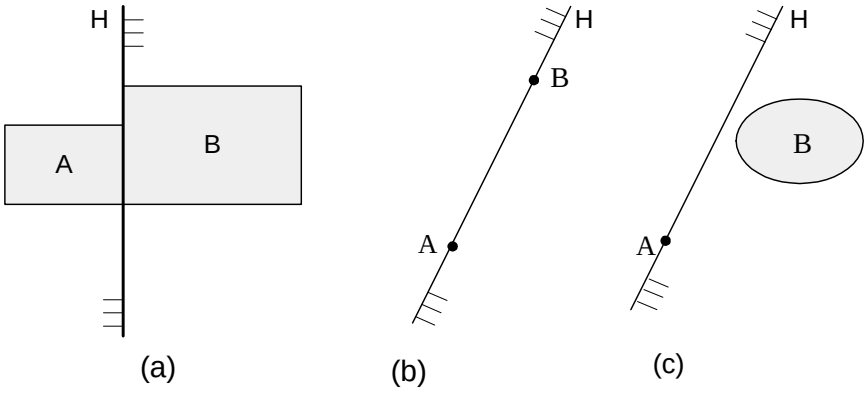


Figura 1.4: Separação dos conjuntos A e B por um hiperplano H.

A separação entre conjuntos nem sempre é possível. Assim, se $A \subset B$ ou $B \subset A$ (ver Figura 1.5), não existe um hiperplano que separe estes conjuntos.

Exemplo 1.3.2 *Considerem-se, em* $\mathbb{R}^2$*, os conjuntos* $A = \{(x, y) : x^2 + y^2 \leq 4\}$ *e* $B = \{(x, y) : x^2 + y^2 \leq 1\}$*. A representação geométrica dos conjuntos* A *e* B *permite-nos concluir que não é possível separar estes conjuntos.*

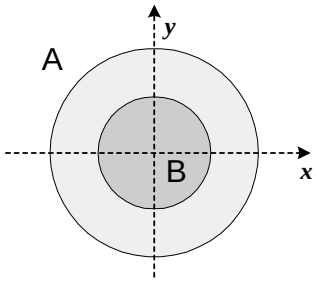


Figura 1.5: Os conjuntos A e B não podem ser separados.

Definição 1.3.3 *Sejam* A *e* B *dois conjuntos e* H *um hiperplano, em* $\mathbb{R}^n$*. Diz-se que* H ***separa estritamente*** *os conjuntos* A *e* B*, se* A *está contido num dos semi-espaços abertos determinados por* H *e* B *está contido no outro semi-espaço aberto. A este tipo de separação dá-se o nome de* ***separação estrita****.*

Na Figura 1.6 podemos observar exemplos de separação estrita.

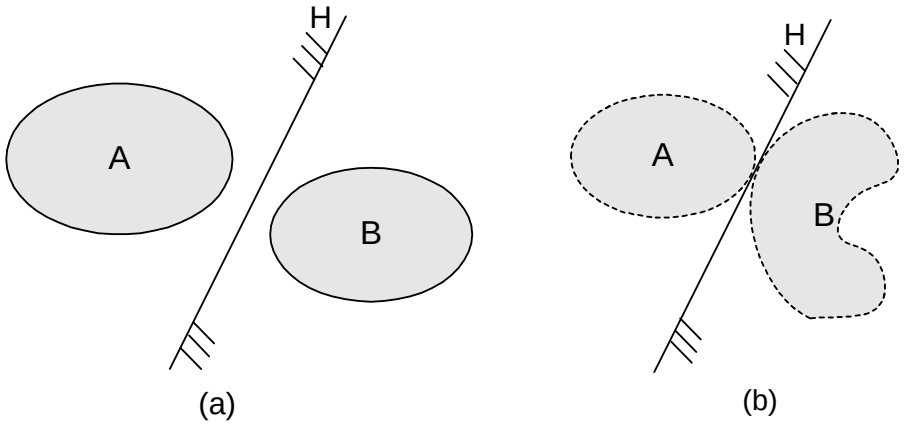


Figura 1.6: Separação estrita entre os conjuntos A e B.

Podemos ainda distinguir um outro tipo de separação entre conjuntos: a *separação própria*.

Definição 1.3.4 *Sejam A e B dois conjuntos e H um hiperplano, em $\mathbb{R}^n$. Diz-se que H* ***separa propriamente*** *os conjuntos A e B, se estes não se encontram simultaneamente contidos em H. A este tipo de separação dá-se o nome de* ***separação própria****.*

Torna-se claro que os exemplos apresentados que na Figura 1.4 (a), 1.4 (c) e na Figura 1.6 são de separação própria, o mesmo não se pode dizer acerca do exemplo apresentado na Figura 1.4 (b).

Exemplo 1.3.5 *Em $\mathbb{R}^2$, sejam $A = \{(x, y) : x > 0, y \geq \frac{1}{x}\}$ e $B = \{(x, y) : x \leq 0, y \in$ $\in \mathbb{R}\}$.*

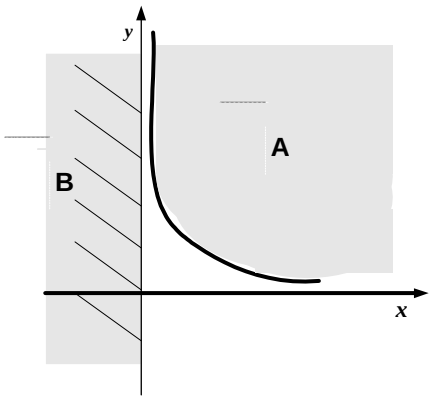


Figura 1.7: Representação geométrica dos conjuntos A e B.

Observando a Figura 1.7, verificamos que não é possível separar estritamente os conjuntos A e B. No entanto, o eixo das ordenadas separa-os e esta separação é própria.

Por vezes, considera-se um outro tipo de separação: a *separação forte*.

Definição 1.3.6 *Sejam S_1 e S_2 dois conjuntos, em $\mathbb{R}^n$. Diz-se que S_1 e S_2 encontram-se* ***fortemente separados*** *por um hiperplano H se existir $r > 0$ tal que $S_1 + rB$ está contido num dos semi-espaços abertos determinados por H e $S_2 + rB$, no outro*

semi-espaço aberto, onde B designa um disco unitário centrado na origem. A este tipo de separação dá-se o nome de ***separação forte****.*

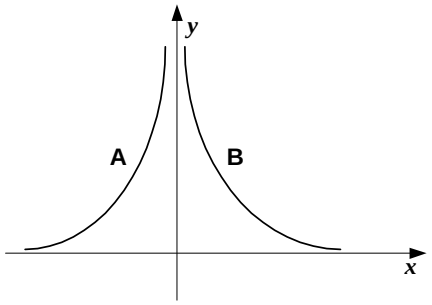


Figura 1.8: Os conjuntos A e B não podem ser fortemente separados.

O conceito de separação estrita é, por vezes, confundido com o de separação forte. Na Figura 1.8 encontramos um exemplo que nos ajuda a distinguir estes dois tipos de separação. Neste caso, os conjuntos A e B encontram-se estritamente separados e não fortemente separados.

Lema 1.3.7 *Sejam S_1 e S_2 dois conjuntos não vazios, em $\mathbb{R}^n$. Para que um hiperplano H separe fortemente os conjuntos S_1 e S_2, é necessário e suficiente que:*

$$\inf\{\|x - y\| : x \in S_1, y \in S_2\} > 0.$$

O Lema 1.3.7 encontra-se demonstrado em [41].

Teorema 1.3.8 *Seja $S \subseteq \mathbb{R}^n$ um conjunto convexo fechado. Então, S é a intersecção de todos os semi-espaços fechados que o contêm.*

Demonstração: Seja S um conjunto não vazio, em $\mathbb{R}^n$, e $a \notin S$. Os conjuntos $S_1 = \{a\}$ e $S_2 = S$ encontram-se nas condições do Lema 1.3.7, ou seja, S_1 e S_2 podem ser fortemente separados por um hiperplano H. Logo, um dos semi-espaços fechados determinados pelo hiperplano H contém o conjunto $S_2 = S$ e não contém S_1. Repetimos este raciocínio para todos os pontos que não pertencem a S. No final, teremos que a

intersecção de todos os semi-espaços obtidos contém o conjunto S, mas não contém nenhum dos pontos que não pertencem a S. $\diamondsuit$

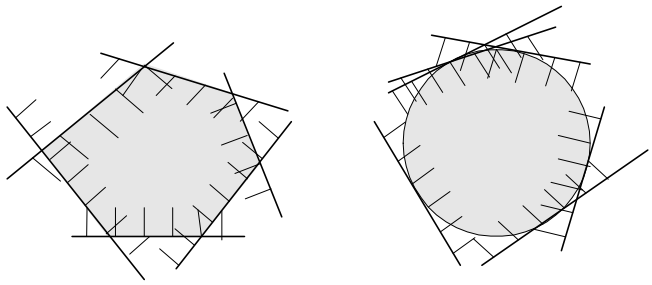


Figura 1.9: Conjuntos convexos, em $\mathbb{R}^2$, vistos como a intersecção de famílias de semi--planos.

No Teorema 1.3.8 quando se fala em intersecção de semi-espaços, fala-se da intersecção de um número finito ou infinito de semi-espaços fechados que contêm S. Na Figura 1.9 podemos observar uma ilustração do Teorema 1.3.8, em $\mathbb{R}^2$.

Lema 1.3.9 *Seja $A \subseteq \mathbb{R}^n$ um conjunto não vazio, fechado e $x \in \mathbb{R}^n$. Então, existe $a_0 \in A$ tal que $d_A(x) = \|x - a_0\|$, onde $d_A(x) = \min_{y \in A} d(x, y)$ é a distância entre o ponto x e o conjunto A.*

Note-se que, em geral, o ponto a_0 pode não ser único.

O Lema 1.3.9 encontra-se demonstrado em [47].

O Lema seguinte permite-nos definir unicamente o ponto de um conjunto convexo mais próximo a um ponto dado.

Lema 1.3.10 *Seja $A \subseteq \mathbb{R}^n$ um conjunto convexo, fechado, não vazio e $x \in \mathbb{R}^n$. Então, existe um único ponto $a_0 \in A$ tal que $\|x - a_0\| = inf\{\|x - z\| : z \in A\}$.*
Mais se afirma, que $\langle x - a_0, a - a_0 \rangle \leq 0$, para cada $a \in A$.

Demonstração: Como A é um conjunto fechado encontra-se nas condições do Lema 1.3.9, que nos garante a existência de um ponto $a_0 \in A$ tal que $\|x - a_o\| = \inf\{\|x - z\| : z \in A\}$.

Seja $a \in A$ e $0 \leq \lambda \leq 1$. Como A é um conjunto convexo, $(1-\lambda)a_0 + \lambda a \ \in A$. Verificamos que qualquer que seja o valor de a_0:

$$\|x - ((1-\lambda)a_0 + \lambda a)\|^2 = \|(x - a_0) + \lambda(a_0 - a)\|^2 \geq \|x - a_0\|^2.$$

Atendendo às propriedades da norma temos que $\langle x - a_0, a - a_0 \rangle \leq 0$.

Suponhamos que $a_1 \in A$. Deste modo, a seguinte igualdade é satisfeita:

$$\|x - a_1\| = \inf\{\|x - z\| : z \in A\}.$$

Assim, temos que:

$$\langle x - a_0, a_1 - a_0 \rangle \leq 0. \tag{1.3.1}$$

Atendendo à simetria entre a_0 e a_1, temos ainda que:

$$\|a_1 - a_0\|^2 = \langle x - a_1, a_0 - a_1 \rangle \leq 0. \tag{1.3.2}$$

De (1.3.1) e (1.3.2) concluímos que $\|a_1 - a_0\|^2 = \langle a_1 - a_0, a_1 - a_0 \rangle \leq 0$, donde $\|a_1 - a_0\| = 0$ e $a_1 = a_0$, o que mostra a unicidade do ponto a_0 de A para x.

$\diamondsuit$

Lema 1.3.11 *Sejam A e B dois conjuntos não vazios, em $\mathbb{R}^n$, onde A é fechado e B é compacto. Então, existem $a_0 \in A$ e $b_0 \in B$ tais que:*

$$\|a_0 - b_0\| = \inf\{\|a - b\| : a \in A, b \in B\}$$

A demonstração do Lema 1.3.11 pode ser encontrada em [47].

Teorema 1.3.12 *Em $\mathbb{R}^n$, sejam A e B dois conjuntos convexos, disjuntos ($A \cap B = \emptyset$), não vazios, onde A é um conjunto fechado e B é compacto. Então, existe um hiperplano H tal que A pode ser estritamente separado de B por H.*

O Teorema 1.3.12 pode ser demonstrado tendo por base o Lema 1.3.10.

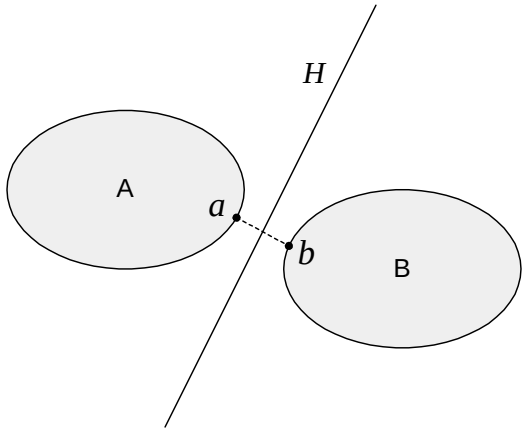


Figura 1.10: Separação entre os conjuntos A e B.

Demonstração: Seja $a \in A$ o ponto do conjunto A cuja distância com qualquer ponto do conjunto B é mínima, e $b \in B$ o ponto do conjunto B cuja distância com qualquer ponto do conjunto A é mínima. Desde que A e B sejam disjuntos, $a \neq b$. Sejam $x \in A$ e $y \in B$. Então, pelo Lema 1.3.10, $\langle b-a, x-a\rangle \leq 0$ e $\langle a-b, y-b\rangle \leq 0$. Assim, $\langle a-b, x\rangle \geq \langle a-b, a\rangle = \frac{1}{2}(\|a\|^2 - \|b\|^2 + \|a-b\|^2) > \frac{1}{2}(\|a\|^2 - \|b\|^2) > \frac{1}{2}(\|a\|^2 - \|b\|^2 - \|a-b\|^2) = \langle a-b, b\rangle \geq \langle a-b, y\rangle$. [2]

Suponhamos que $\vec{c} := a-b$ e $c_0 := \frac{1}{2}(\|a\|^2 - \|b\|^2)$. Então, fica provado que o hiperplano $H = \{\vec{x} \in \mathbb{R}^n : \langle \vec{c}, \vec{z}\rangle = c_0\}$ separa estritamente A e B. $\diamondsuit$

O Exemplo 1.3.5 mostra que no Teorema 1.3.12 o facto de pelo menos um dos conjuntos A e B ser compacto é essencial.

Corolário 1.3.13 *Em $\mathbb{R}^n$, seja S um conjunto convexo fechado e b um ponto tal que $b \notin S$. Então, S e $\{b\}$ podem ser estritamente separados por um hiperplano, em $\mathbb{R}^n$.*

A demonstração do Corolário 1.3.13 encontra-se em [47].

[2] A igualdade $\langle (a-b), a\rangle = \frac{1}{2}(\|a\|^2 - \|b\|^2 - \|a-b\|^2)$ obtém-se, facilmente, tendo em consideração que $\|a-b\|^2 = \|a\|^2 + \|b\|^2 - 2\langle b, a\rangle$.

Teorema 1.3.14 *Em $\mathbb{R}^n$, sejam S_1 e S_2 conjuntos convexos. Então, $0 \in ir(S_1 - S_2)$[3] se e só se $ir\,(S_1) \cap ir\,(S_2) \neq \emptyset$.*

Lema 1.3.15 *Em $\mathbb{R}^n$, seja S um conjunto convexo, não vazio, que não contém a origem. Então, existe um hiperplano H que não contém S e que separa a origem do conjunto S.*

O Teorema 1.3.14 encontra-se demonstrado em [18] e o Lema 1.3.15 em [47].

Teorema 1.3.16 *Em $\mathbb{R}^n$, sejam A e B dois conjuntos convexos, não vazios e tais que $ir\,(A) \cap ir(B) = \emptyset$. Então, A e B podem ser separados propriamente por um hiperplano H.*

Demonstração: Considere-se o conjunto $A - B$ que é convexo e não vazio. Pelo Teorema 1.3.14 o conjunto $A - B$ não contém a origem. Logo, pelo Lema 1.3.15, existe um hiperplano, em $\mathbb{R}^n$, que separa $\{0\}$ e $A - B$ e que não contém $A - B$. Assim, existe um $\vec{c} \in \mathbb{R}^n$, com $\vec{c} \neq 0$, e $c_0 \in \mathbb{R}$ tal que:

$$0 = \langle \vec{c}, \vec{0} \rangle \leq c_0 \quad \text{e} \quad \langle \vec{c}, a - b \rangle \geq c_0, \text{ para cada } a \in A \text{ e } b \in B$$

e, para algum $a_0 \in A$ e $b_0 \in B$, temos que:

$$\langle \vec{c}, a_0 - b_0 \rangle > c_0 \geq 0.$$

Para todo $a \in A$ e $b \in B$, temos ainda que $\langle \vec{c}, \vec{a} \rangle \geq \langle \vec{c}, \vec{b} \rangle + c_0 \geq \langle \vec{c}, \vec{b} \rangle$, uma vez que $c_0 \geq 0$. Assim, existe um escalar d satisfazendo as desigualdades:

$$inf\{\langle \vec{c}, \vec{a} \rangle : \vec{a} \in A\} \geq d \geq \sup\{\langle \vec{c}, \vec{b} \rangle : \vec{b} \in B\}.$$

Para cada $\vec{a'} \in A$ e $\vec{b'} \in B$ tem-se que $\langle \vec{c}, \vec{a'} \rangle \geq d \geq \langle \vec{c}, \vec{b'} \rangle$, donde se conclui que o hiperplano H de equação $\langle \vec{c}, \vec{z} \rangle = d$ separa os conjuntos A e B. Assim, H não pode conter simultaneamente A e B, pois se tal ocorresse $\langle \vec{c}, a_0 - b_0 \rangle = 0$, o que contradiz o Lema 1.3.10. Conclui-se, assim, que H separa A e B propriamente. $\diamondsuit$

[3] $S_1 - S_2 := S_1 + (-S_2)$, de acordo com as operações definidas na Definição 1.2.13.

Um conceito que se encontra associado ao conceito de separação é o de *suporte*.

Definição 1.3.17 *Seja $S \subseteq \mathbb{R}^n$. Um semi-espaço, em $\mathbb{R}^n$, que suporta S é um semi - - espaço fechado que contém S e que contém uma parte de S na sua fronteira.*

O conceito de suporte pode ser adaptado aos hiperplanos.

Definição 1.3.18 *Dizemos que um hiperplano H suporta o conjunto S (ou que H é um hiperplano suporte de S), se S estiver contido num dos semi-espaços fechados determinados por H e se $S \cap H \neq \emptyset$.*

Na Figura 1.11 podemos observar vários semi-espaços fechados, em $\mathbb{R}^2$, determinados pelos hiperplanos $H_1, ..., H_8$, que contêm o conjunto $S \subseteq \mathbb{R}^2$. No entanto, só os hiperplanos $H_1, ..., H_6$ são hiperplanos suporte.

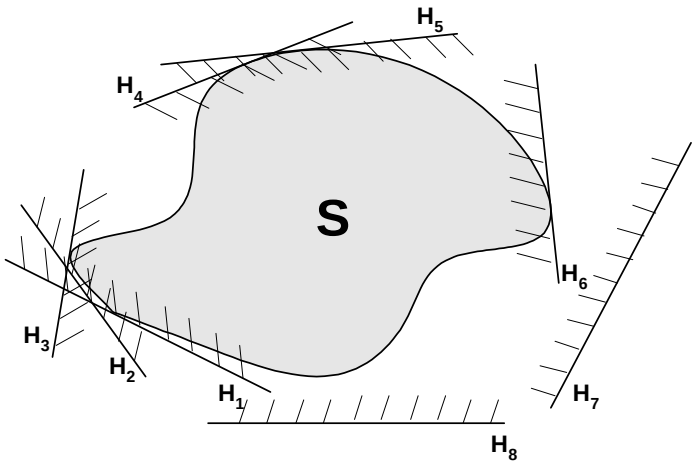


Figura 1.11: Hiperplanos de suporte em vários pontos do conjunto S, em $\mathbb{R}^2$.

Teorema 1.3.19 *Se $S \subseteq \mathbb{R}^n$ é um conjunto convexo, então por cada ponto da fronteira de S passa pelo menos um hiperplano suporte.*

O Teorema 1.3.19 encontra-se demonstrado em [22].

Definição 1.3.20 *Um* ***hiperplano*** *suporte H é* ***tangente*** *a um conjunto convexo fechado S num ponto $x \in S$, se H for o único hiperplano suporte de S em x.*

Definição 1.3.21 *Seja S um conjunto convexo em $\mathbb{R}^n$. Um* ***semi-espaço tangente*** *ao conjunto S é um semi-espaço limitado por um hiperplano tangente a S nalgum ponto $x \in S$.*

Teorema 1.3.22 *Seja S um conjunto convexo fechado de $\mathbb{R}^n$. Então, S é a intersecção dos semi-espaços fechados tangentes que o contém.*

O Teorema 1.3.22 encontra-se demonstrado em [41].

Observação 1.3.23 *Em $\mathbb{R}^2$, aos hiperplanos (rectas) suporte chamamos* ***arestas suporte*** *e aos pontos de intersecção das arestas suporte do conjunto S ($S \subseteq \mathbb{R}^2$) com S ($S \subseteq \mathbb{R}^2$) chamamos* ***vértices suporte****.*

Capítulo 2

Invólucros Convexos

2.1 Invólucro Convexo: definições e propriedades básicas

Nesta secção, abordam-se um conjunto de definições e resultados que se revelam de grande interesse no estudo do capítulo seguinte, sobre algoritmos que determinam invólucros convexos de conjuntos de pontos nos espaços bidimensional e tridimensional.

Definição 2.1.1 *Chama-se* ***cadeia poligonal*** *a um conjunto de n pontos distintos do plano* $v_1, v_2, ..., v_n$*, chamados vértices, ligados por segmentos de recta* $[v_1, v_2], [v_2, v_3], ..., [v_{n-1}, v_n]$*, chamados arestas.*

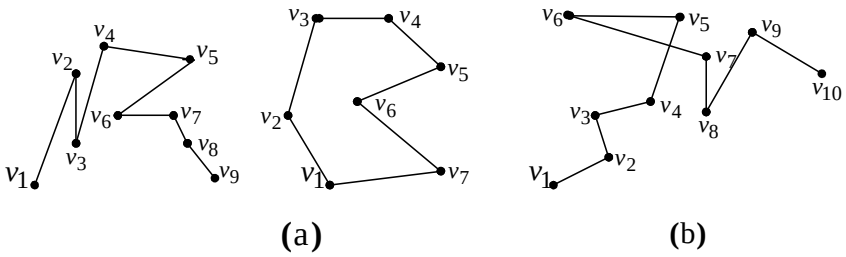


Figura 2.1: (a) Cadeias poligonais simples ; (b) Cadeia poligonal não simples.

Caso não exista intersecção entre as arestas não adjacentes que constituem a cadeia poligonal estamos perante uma **cadeia poligonal simples** (ver, por exemplo, Figura 2.1 (a)).

Definição 2.1.2 *Uma **cadeia poligonal** diz-se **fechada** se o vértice inicial coincidir com o vértice final.*

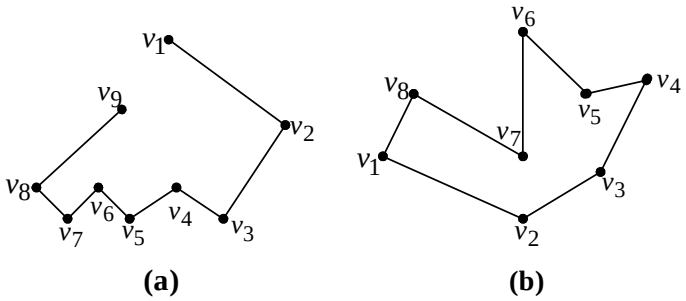


Figura 2.2: (a) Cadeia poligonal aberta ; (b) Cadeia poligonal fechada.

Definição 2.1.3 *Chama-se **polígono simples** P ao conjunto dos pontos de $\mathbb{R}^2$ limitados por uma cadeia poligonal simples fechada reunidos com os pontos da cadeia poligonal simples fechada.*

Como se pode observar na Figura 2.3 qualquer cadeia poligonal simples fechada divide o plano em duas regiões distintas, uma interior à fronteira do polígono e a outra situada na parte exterior.

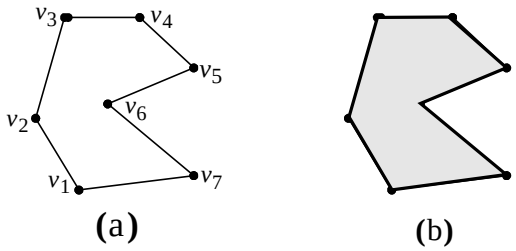


Figura 2.3: (a) Cadeia poligonal simples fechada ; (b) Polígono simples.

Definição 2.1.4 *Um **polígono** diz-se **regular** se em cada vértice tiver os ângulos formados pelas arestas adjacentes iguais e se as arestas tiverem o mesmo comprimento.*

A determinação do menor polígono convexo contendo a totalidade dos pontos de um conjunto constitui, como já foi referido, um dos problemas centrais em *Geometria Computacional*. Este problema consiste na determinação do *invólucro convexo* de um dado conjunto de pontos.

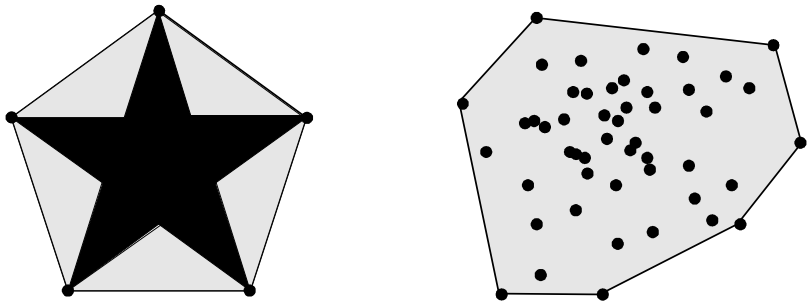


Figura 2.4: Invólucros Convexos.

Definição 2.1.5 *Seja S um subconjunto, não vazio, de $\mathbb{R}^n$. Chama-se* ***invólucro convexo*** *de S e denota-se por $CH(S)$*[1]*, à intersecção de todos os conjuntos convexos que contêm S.*

Como se pode observar na Figura 2.4, o invólucro convexo de S pode também ser entendido como o menor (por inclusão) conjunto convexo contendo S, o que significa que $CH(S)$ é o conjunto convexo que contém todos os pontos de S e não existe mais nenhum conjunto convexo C com esta propriedade, e tal que $CH(S) \subset C$.

Teorema 2.1.6 *Um conjunto $S \subseteq \mathbb{R}^n$ é um conjunto convexo se e só se $CH(S) = S$.*

Demonstração: Seja S um conjunto convexo.
Por definição de invólucro convexo e tendo em conta o resultado do Teorema 1.2.12, fica claro que $CH(S) \supseteq S$.
Seja C um conjunto convexo tal que $S \subset C$. Deste modo, $C \supseteq CH(S)$. Além disso, $S \supseteq CH(S)$ uma vez que S é um conjunto convexo e $S \subseteq C$. Donde se conclui que $S = CH(S)$.

[1]da expressão em Inglês: *Convex Hull*

Suponha-se, agora, que S não é um conjunto convexo.
Por definição, sabemos que o invólucro convexo de um conjunto S é a intersecção de todos os conjuntos convexos que contêm S. Assim, $CH(S)$ será necessariamente um conjunto convexo (ver Teorema 1.2.8). Donde se conclui que $S \neq CH(S)$. $\diamondsuit$

Teorema 2.1.7 *Em $\mathbb{R}^n$, o invólucro convexo de um conjunto aberto é um conjunto aberto.*

Demonstração: Seja C um conjunto aberto, em $\mathbb{R}^n$. Se $x \in CH(C)$, então x é combinação convexa de algum subconjunto finito de elementos de C, isto é, $x = \lambda_1 c_1 + +... + \lambda_m c_m$, para alguns $c_1, ..., c_m \in C$ e alguns $\lambda_1, ..., \lambda_m \geq 0$, com $\sum_{i=1}^{m} \lambda_i = 1$. Como C é, por hipótese, um conjunto aberto existem $r_1, ..., r_m > 0$ tais que $B(c_1; r_1) \subseteq \subseteq C, ..., B(c_m; r_m) \subseteq C$. Considerando $r = min\{r_1, ...r_m\}$, $r > 0$, vamos demonstrar que $B(x; r) \subseteq CH(C)$. Para cada $y \in B(x; r)$, a desigualdade $\|y - x\| < r$ é satisfeita. Para cada $i = 1, ..., m$, o ponto $x_i := c_i + y - x$ está na bola $B(c_i; r)$ e, consequentemente, na bola $B(c_i; r_i)$, $\forall i = 1, ..., m$. Mas, a bola $B(c_i; r_i) \subseteq C$, logo $x_i \in C$. Pela construção, $y = x_i + x - c_i$, onde $x_i \in C$, $x \in C$ e $c_i \in C, \forall i = 1, ..., m$. Então, tem-se que $y \in C$ e fica provado que $\forall x \in C$, $\exists r > 0 : B(x; r) \subset C$. Uma vez que $C \subset CH(C)$, $B(x; r) \subset CH(C)$ e o teorema está demonstrado. $\diamondsuit$

Teorema 2.1.8 *Em $\mathbb{R}^n$, o invólucro convexo de um conjunto compacto é um conjunto compacto.*

Corolário 2.1.9 *O invólucro convexo de um conjunto finito de pontos, em $\mathbb{R}^n$, é um conjunto compacto.*

O Teorema 2.1.8 encontra-se demonstrado em [10] e o Corolário 2.1.9 em [47].

A proposição seguinte ajuda a perceber a estrutura dos invólucros convexos de um conjunto finito de pontos em $\mathbb{R}^2$ e em $\mathbb{R}^n$.

Proposição 2.1.10 *Seja S um conjunto finito de pontos, em $\mathbb{R}^n$. O invólucro convexo de S pode ser definido por qualquer uma das seguintes expressões:*

i) O invólucro convexo de S, em $\mathbb{R}^n$, é o conjunto de todas as combinações convexas dos pontos de S, ou seja, $CH(S) = \{\alpha_1 p_1 + ... + \alpha_m p_m : \alpha_1 + ... + \alpha_m = 1$, com $\alpha_i \geq 0,\ i = 1, ..., m\}$, $\forall m \in \mathbb{N}, m \leq |S|$.

ii) O invólucro convexo de S, em $\mathbb{R}^n$, é o conjunto de todas as combinações convexas de quaisquer $n + 1$ pontos de S.

Como se verá mais adiante, a afirmação *ii)* é baseada no *Teorema de Caratheodory* (Teorema 2.2.2) e distingue-se do apresentado na alínea *i)* apenas no que diz respeito ao número de pontos usados pelo que, neste caso, será exactamente de $n+1$ pontos. Se $S \subset \mathbb{R}^2$, por exemplo, então o $CH(S)$ é o conjunto de todas as combinações convexas dos seus subconjuntos de três pontos. De salientar, que caso esses três pontos não sejam colineares eles formam triângulos, como podemos observar na Figura 2.5.

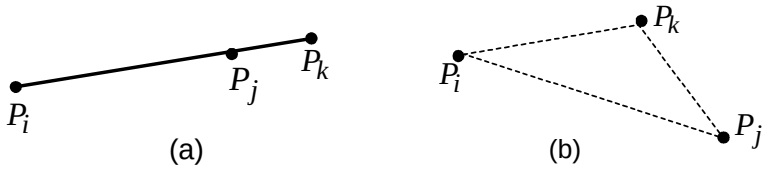


Figura 2.5: (a) três pontos colineares ; (b) três pontos não colineraes.

iii) O invólucro convexo de S, em $\mathbb{R}^n$, é a intersecção de todos os semi-espaços que contêm S.

Note-se que, em $\mathbb{R}^2$, um semi-espaço corresponde a um semi-plano. Assim, podemos pensar no $CH(S)$, $S \subseteq \mathbb{R}^2$, como sendo uma intersecção de semi-planos.

iv) O invólucro convexo de S, em $\mathbb{R}^2$, é o menor polígono convexo P que contém S. Neste sentido, não existirá outro polígono P' tal que: $S \subseteq P' \subset P$.

v) O invólucro convexo de S, em $\mathbb{R}^2$, é o polígono convexo de menor área que contém S.

vi) O invólucro convexo de S, em $\mathbb{R}^2$, é o polígono convexo de menor perímetro que contém S.

vii) O invólucro convexo de S, em $\mathbb{R}^2$, é a união de todos os triângulos determinados por pontos de S.

2.2 Representação de conjuntos convexos

Em seguida, apresentam-se alguns resultados importantes sobre representação de conjuntos convexos em geral (*Teorema de Caratheodory*), e sobre representação de conjuntos convexos finitos, em particular.

O *Teorema de Caratheodory* resulta da definição de invólucro convexo e assume um papel central na representação de conjuntos convexos. Este teorema tem por base a ideia de que, se um dado ponto s de $\mathbb{R}^n$ pertencer ao invólucro convexo de um conjunto de pontos S, então existirá um subconjunto finito $S^{'}$ de S contendo no máximo $n+1$ pontos tais que s está no invólucro convexo do conjunto S.

Exemplo 2.2.1 *Considere-se, em $\mathbb{R}^2$, o conjunto $S = \{(0,0); (2,0); (3,2); (1,2)\}$.*

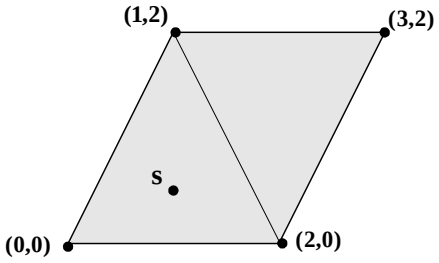


Figura 2.6: Ilustração do Teorema de Caratheodory relativamente ao conjunto S.

O invólucro convexo do conjunto S é um paralelogramo, como mostra a Figura 2.6. No entanto, se se considerar um ponto $s = (1, 1/2)$, localizado no interior do $CH(S)$

é possível encontrar um subconjunto de S, com $2+1=3$ pontos, por exemplo, $S' = = \{(0,0);(2,0);(1,2)\}$, cujo invólucro convexo contém o ponto s.

Teorema 2.2.2 (Caratheodory) *Seja S um qualquer conjunto, em $\mathbb{R}^n$, e seja $s \in \in CH(S)$. Então, s pode ser escrito como combinação convexa de $n+1$ elementos de S.*

Demonstração: Seja S um conjunto em $\mathbb{R}^n$ e $s \in CH(S)$. Pela Proposição 2.1.10 *i)* existem m pontos $s_1, ..., s_m$ de S e constantes $\lambda_1, ..., \lambda_m \geq 0$, com $\lambda_1 + ... + \lambda_m = 1$, tais que $s = \lambda_1 s_1 + ... + \lambda_m s_m$.
Admitamos, agora, que $s_1, ..., s_m$ foram escolhidos de forma a que s não possa ser escrito como combinação convexa de menos de m pontos de S. Pretendemos mostrar que $m \leq n+1$.
Suponhamos, por contradição, que $m > n+1$. Então, desde que S tenha dimensão menor ou igual a $n+1$, os m pontos $s_1, ..., s_m$ são dependentes, ou seja, existem m constantes $\mu_1, ..., \mu_m$, que não são simultaneamente nulas, tais que $\mu_1 + ... + \mu_m = 0$ e $\mu_1 s_1 + ... + \mu_m s_m = 0$. Seja $t > 0$ tal que as constantes $\lambda_1 + \mu_1 t, ..., \lambda_m + \mu_m t$ não são negativas e em que pelo menos uma delas é nula. Este t existirá desde que $\lambda_i \geq 0$, $i = 1, \ldots, m$, e que pelo menos um dos μ_i, $i = 1, ..., m$, seja negativo. Suponhamos, sem perda de generalidade, que $\lambda_m + \mu_m t = 0$. Neste caso, $s = (\lambda_1 + \mu_1 t)s_1 + ... + (\lambda_{m-1} + \mu_{m-1} t)s_{m-1}$, o que contradiz a minimalidade de m, mostrando que $m \leq n+1$. $\diamondsuit$

O *Teorema de Radon*, apresentado em seguida, é interessante uma vez que para além de ser um auxílio no estudo de outros resultados, ele permite saber a possível disposição de quatro pontos no plano.

Teorema 2.2.3 (Radon) *Sejam $s_1, ..., s_m \in \mathbb{R}^n$, com $m \geq n+2$. Então, o conjunto $\{s_1, ..., s_m\}$ pode ser dividido em dois subconjuntos U e V, $\{s_1, ..., s_m\} = U \cup V$, $U \cap V = = \emptyset$, tais que $CH(U) \cap CH(V) \neq \emptyset$.*

O Teorema 2.2.3 encontra-se demonstrado em [47].

A Figura 2.7 ilustra duas possibilidades quanto à localização de quatro pontos distintos no plano. Assim, ou estes pontos formam um quadrilátero (b) ou um dos pontos encontra-se no interior de um triângulo definido pelos restantes três pontos (a). No que se refere à aplicação do Teorema 2.2.3 aos conjuntos representados na Figura 2.7, no caso (a), $U = \{d\}$ e $V = \{a, b, c\}$, no caso (b), $U = \emptyset$ e $V = \{a, b, c, d\}$.

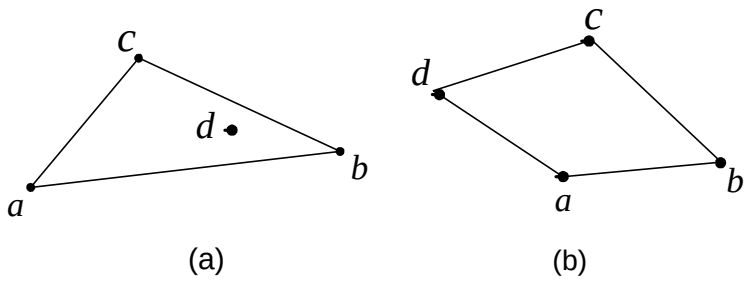


Figura 2.7: Ilustração do Teorema Radon.

2.3 Pontos extremos, faces e facetas

Definição 2.3.1 *Seja $S \subseteq \mathbb{R}^n$. Diz-se que $p \in S$ é um **ponto extremo** de S, se p não pode ser escrito como uma combinação convexa dos elementos de S distintos de p.*

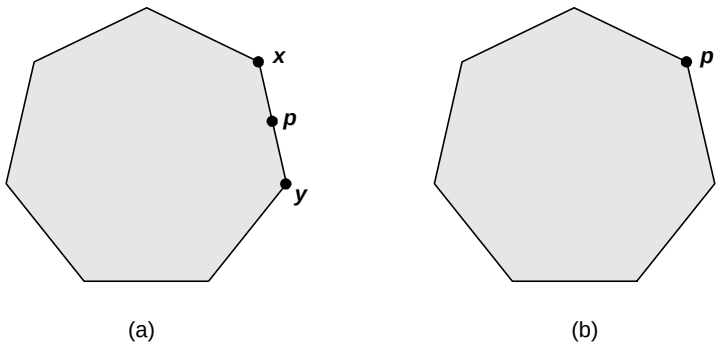


Figura 2.8: (a) p não é ponto extremo; (b) p é ponto extremo.

Assim, um ponto $p \in S$ será um *ponto extremo* de um subconjunto S de $\mathbb{R}^2$, se não existirem dois pontos $x, y \in S$, $x \neq p$ e $y \neq p$, tais que $p \in [x, y]$.

Em $\mathbb{R}^2$, a verificação de que um ponto p não é ponto extremo pode ainda ser feita recorrendo ao lema seguinte.

Lema 2.3.2 *Seja S um conjunto finito de pontos, em $\mathbb{R}^2$. Um ponto $p \in S$ não é um ponto extremo se e só se ele se encontrar no interior de um triângulo cujos vértices são pontos de S distintos de p* (ver Figura 2.9).

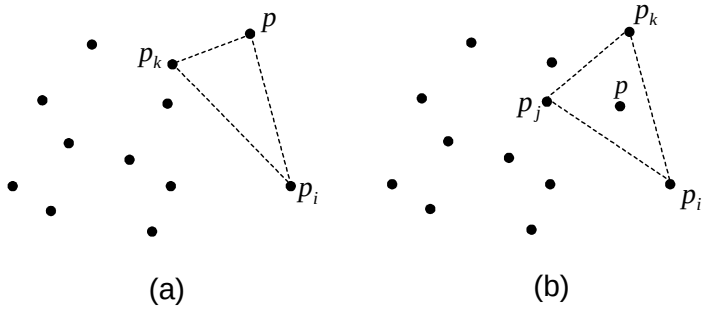


Figura 2.9: (a) p é ponto extremo; (b) p não é ponto extremo.

A demonstração do Lema 2.3.2 é imediata atendendo ao resultado da Proposição 2.1.10 *vii*) e à definição de ponto extremo. Pode ser encontrada em [38].

Recorrendo à Definição 2.3.1 é fácil verificar que o invólucro convexo de um conjunto finito de pontos S fica completamente caracterizado pelo invólucro convexo do conjunto dos seus pontos extremos $ext\,(S)$. Neste sentido, $CH(ext\,(S)) = CH(S)$. Combinando este resultado com a definição 2.1.10 *ii*), é possível concluir que se $S \subseteq \mathbb{R}^n$, então qualquer elemento de S é uma combinação convexa de, no máximo, $n + 1$ pontos extremos de S.

Teorema 2.3.3 *Seja S um conjunto convexo, em $\mathbb{R}^n$. Um ponto $p \in S$ é um ponto extremo de S se e só se $S \setminus \{p\}$ for convexo.*

Demonstração: Seja S um conjunto convexo, não vazio, de $\mathbb{R}^n$ e p um dos seus pontos.

($\Rightarrow$) (Por contradição) Suponhamos que $p \in S$ é um ponto extremo, mas $S\backslash\{p\}$ não é um conjunto convexo. Então, existem $x, y \in S\backslash\{p\}$, com $x \neq y$, tais que o ponto $(1-\lambda)x + \lambda y \notin S\backslash\{p\}$, para algum $\lambda \in]0,1[$. Como p é ponto extremo de S, p não poderá ser escrito como combinação convexa dos pontos de $S\backslash\{p\}$. Logo, para algum $\lambda \in]0,1[$ temos que $(1-\lambda)x + \lambda y \notin S$, o que contradiz a convexidade do conjunto S.
($\Leftarrow$) Reciprocamente, se $S\backslash\{p\}$ for um conjunto convexo, dados $x, y \in S\backslash\{p\}$ quaisquer, o segmento $[x, y]$ é um subconjunto de $S\backslash\{p\}$, o que faz com que não existam dois pontos $x', y' \in S$, com $x' \neq y'$, $x' \neq p$ e $y' \neq p$, de tal forma que $p \in]x', y'[$, logo p é ponto extremo. $\diamondsuit$

Teorema 2.3.4 *Seja S um conjunto convexo, compacto, não vazio, de $\mathbb{R}^n$. Então, S admite um ponto extremo.*

O Teorema 2.3.4 encontra-se demonstrado, por exemplo, em [42].

Definição 2.3.5 *Chama-se **face** de um conjunto convexo S de $\mathbb{R}^n$ a todo o subconjunto F de S de tal forma que, sempre que $\lambda x + (1-\lambda)y \in F$, onde $x, y \in S$ e $\lambda \in]0,1[$, então $x, y \in F$.*

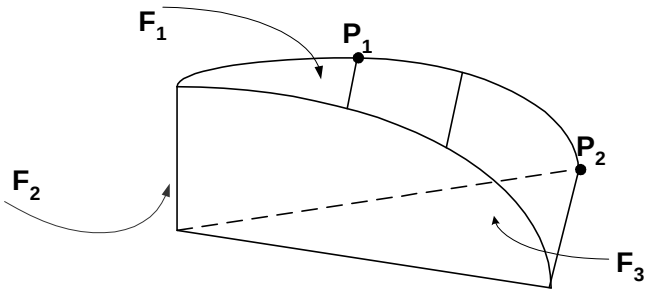


Figura 2.10: Faces e pontos extremos.

Na Figura 2.10, F_1 representa uma face de dimensão dois e F_2 uma face de dimensão um. Quanto a P_1 e P_2, representam pontos extremos.

Teorema 2.3.6 *Seja S um conjunto convexo de $\mathbb{R}^n$. Então:*

i) a intersecção de qualquer família, não vazia, de faces de S é uma face de S;

ii) se F é uma face de S e F' é uma face de F, então F' é uma face de S;

iii) a intersecção de S com cada um dos seus hiperplanos suporte é uma face de S.

Demonstração:

i) Seja $(F_i, i \in I)$ uma família, não vazia, de faces de S. Então, $\bigcap_{i\in I} F_i$ é um subconjunto de S. Já agora, $\lambda x + (1-\lambda)y \in \bigcap_{i\in I} F_i$, com $x, y \in S$, $\lambda \in]0,1[$. Logo, $\forall i \in I$ tem-se $\lambda x + (1-\lambda)y \in F_i$ e, uma vez que F_i é uma face tem-se que $x, y \in F_i$, $\forall i \in I$. Consequentemente, $x, y \in \bigcap F_i$, o que demonstra que $\bigcap F_i$ é uma face de S.

ii) Seja F uma face de um conjunto convexo S e seja F' uma face de F. Se $\lambda x+(1-\lambda)y \in$ $\in F'$, onde $x, y \in S$ e $\lambda \in]0,1[$, então $\lambda x + (1-\lambda)y \in F$, desde que $F' \subseteq F$. Uma vez que F é uma face de S, então $x, y \in F$. Mas, F' é uma face de F, o que significa que se $\lambda x + (1-\lambda)y \in F'$, com $x, y \in F$, logo $x, y \in F'$. Assim, obtemos que se $\lambda x + (1-\lambda)y \in F'$, com $x, y \in S$ e $\lambda \in]0,1[$, então $x, y \in F'$ e fica provado que F' é uma face de S.

iii) Seja $H = \{\vec{x} : \langle \vec{c}, \vec{a} \rangle = \alpha\}$ um hiperplano de suporte para S, onde $\alpha \in \mathbb{R}$, $\vec{x} \in \mathbb{R}^n$ e $\vec{x} \neq 0$. Suponhamos que $\langle \vec{c}, \vec{b} \rangle \leq \alpha$, sempre que $\vec{b} \in S$. Se $\lambda\vec{x} + (1-\lambda)\vec{y} \in S \cap H$, onde $\vec{x}, \vec{y} \in S$ e $\lambda \in]0,1[$, então $\alpha = \langle \vec{c}, \lambda\vec{x} + (1-\lambda)\vec{y} \rangle = \lambda\langle \vec{c}, \vec{x} \rangle + (1-\lambda)\langle \vec{c}, \vec{y} \rangle \leq$ $\leq \lambda\alpha + (1-\lambda)\alpha = \alpha$.
Desde que $\lambda \geq 0$ temos que $\langle \vec{c}, \vec{x} \rangle = \langle \vec{c}, \vec{y} \rangle = \alpha$, resultando que $\vec{x}, \vec{y} \in S \cap H$. Logo, $S \cap H$ é uma face de S. $\diamondsuit$

Definição 2.3.7 *Seja S um conjunto convexo, não vazio. Chama-se* ***face exposta*** *de S a toda a face que resulta da intersecção de S com um hiperplano suporte de S. A cada face exposta cuja dimensão é zero chama-se* ***ponto exposto****.*

Na Figura 2.10, F_2 e F_3 são exemplos de faces expostas, o mesmo não acontece com F_1.

Definição 2.3.8 *A uma função $f : V \to \mathbb{K}$, onde V é um espaço vectorial sobre um corpo $\mathbb{K}$ chama-se* ***funcional linear*** *se para $\forall \vec{u}, \vec{v} \in V$ e $\forall \lambda \in \mathbb{K}$ é satisfeito:*

i) $f(\vec{u} + \vec{v}) = f(\vec{u}) + f(\vec{v})$;

ii) $f(\lambda \vec{v}) = \lambda f(\vec{v})$.

Exemplo 2.3.9 *O produto escalar $\langle \vec{x}, \vec{a} \rangle$ é uma funcional linear, para $\vec{x}, \vec{a} \in \mathbb{R}^n$.*

Lema 2.3.10 *Em $\mathbb{R}^n$, seja f uma funcional linear contínua e seja S um conjunto não vazio, convexo e compacto. Então, o conjunto de pontos onde f atinge o seu mínimo (respectivamente, máximo) em S é uma face de S.*

A prova do Lema 2.3.10 pode ser encontrada em [42].

Teorema 2.3.11 (Krein-Milman) *Seja S um conjunto não vazio, convexo e compacto, em $\mathbb{R}^n$, e P o conjunto dos seus pontos extremos. Então, $S = \overline{CH(P)}$.*

Demonstração: Seja P o conjunto dos pontos extremos de S. Dado que S é um conjunto não vazio, convexo e compacto, tem-se

$$\overline{CH(P)} \subset S.$$

Assim, para mostrar o teorema, basta mostrar que $S \subset \overline{CH(P)}$. Suponhamos que existe $s_0 \in S \setminus \overline{CH(P)}$. Segundo o Corolário 1.3.13 é possível separar estritamente $\overline{CH(P)}$ e s_0 por um hiperplano H.

Seja H um hiperplano de equação $\langle \vec{x}, \vec{a} \rangle = \alpha$, assuma-se que $\langle \vec{s_0}, \vec{a} \rangle < \alpha$, onde $\vec{s_0} = = 0s_0$, $\vec{a} \in \mathbb{R}^n$, e que o conjunto $\overline{CH(P)}$ se encontra contido no semi-espaço determinado pela desigualdade $\langle \vec{x}, \vec{a} \rangle \geq \alpha$. Então, para todo $\vec{p} \in \overline{CH(P)}$ temos que $\langle \vec{s_0}, \vec{a} \rangle < \langle \vec{p}, \vec{a} \rangle$. Considere-se, agora, o conjunto $S' = \{\vec{x} \in S : \langle \vec{x}, \vec{a} \rangle = \min_{\vec{s} \in S} \langle \vec{s}, \vec{a} \rangle\}$. Pelo Lema 2.3.10, S' é uma face de S e, pelo Teorema 2.3.4, S' contém um ponto extremo de S, ou seja, $P \cap C' \neq \emptyset$. Logo, $\overline{CH(P)} \cap S' \neq \emptyset$. Resultando que $S \subset \overline{CH(P)}$. $\diamondsuit$

Straszewicz utilizando o *Teorema de Krein-Milman* e conceito de ponto exposto, por ele introduzido, apresentou um teorema que mostra que qualquer conjunto convexo e compacto é o invólucro convexo fechado dos seus pontos expostos. Este resultado foi, mais tarde, provado por Klee [32] para subconjuntos não vazios, convexos e compactos de um espaço normado de dimensão finita.

Lema 2.3.12 *Um conjunto compacto, não vazio, em $\mathbb{R}^n$, tem um ponto exposto no semi-espaço fechado que o contém.*

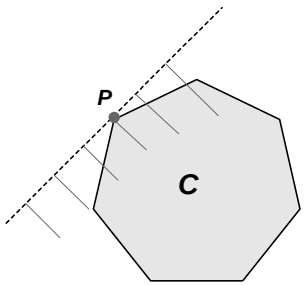


Figura 2.11: O ponto P encontra-se no semi-espaço fechado que contém o conjunto C.

O Lema 2.3.12 encontra-se demonstrado em [47].

Teorema 2.3.13 (Teorema de Straszewicz) *Todo o conjunto não vazio, convexo e compacto, em $\mathbb{R}^n$, é o invólucro convexo fechado dos seus pontos expostos.*

Demonstração: Seja A um conjunto não vazio, convexo e compacto, em $\mathbb{R}^n$, e seja B o conjunto dos seus pontos expostos. Como B é o conjunto dos pontos expostos de A, sabemos que $\overline{CH(B)} \subseteq A$. Logo, basta-nos mostrar que $A \subseteq \overline{CH(B)}$.

Suponhamos, por redução ao absurdo, que existe um ponto x_0 de A tal que $x_0 \notin \overline{CH(B)}$. Assim, pelo Teorema 1.3.12, existe um semi-espaço aberto E em $\mathbb{R}^n$, disjunto de $\overline{CH(B)}$ e contendo x_0. Pelo Lema 2.3.12, o ponto x_0 é um ponto exposto. Deste modo, $x_0 \in B$ e, consequentemente, $x_0 \in \overline{CH(B)}$. A contradição prova que $A \subseteq \overline{CH(B)}$. $\diamondsuit$

2.4 Poliedros e Polítopos

Nesta secção, será feita uma breve abordagem aos poliedros e aos polítopos. Para tal, utilizaremos alguns conceitos e propriedades estudados em secções anteriores. Esta secção terá uma maior aplicabilidade no capítulo seguinte, quanto forem apresentados algoritmos que constroem invólucros convexos em $\mathbb{R}^3$.

Definição 2.4.1 *Um conjunto P, em $\mathbb{R}^n$, é um **polítopo** se for o invólucro convexo de um conjunto finito de pontos, em $\mathbb{R}^n$.*

Como o invólucro convexo de um conjunto finito de pontos é um conjunto compacto e como um polítopo é o invólucro convexo de um conjunto finito de pontos, podemos concluir que um polítopo é um conjunto compacto.

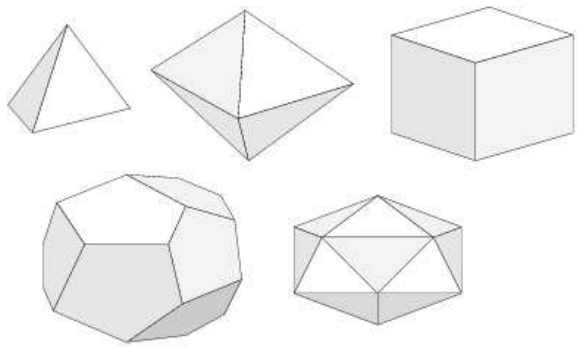


Figura 2.12: Sólidos Platónicos: tetraedro, octaedro, cubo, dodecaedro e icosaedro.

Definição 2.4.2 *Um **polítopo regular** é um polítopo cujas faces são polígonos regulares congruentes e em que o número de faces incidentes em cada vértice é o mesmo.*

Em $\mathbb{R}^3$, existem apenas cinco polítopos regulares, os quais são conhecidos como os *Sólidos de Platão*: o tetraedro, o cubo, o octaedro, o dodecaedro e o icosaedro (ver Figura 2.12).

Polítopo Regular	**V**	**A**	**F**
Tetraedro	4	6	4
Cubo	8	12	6
Octaedro	6	12	8
Dodecaedro	20	30	12
Icosaedro	12	30	20

A famosa Fómula de Euler estabelece uma relação importante entre os vértices (V), as arestas (A) e as faces (F) de um polítopo regular.

Teorema 2.4.3 (Fórmula de Euler) *Sejam V, A e F o número de vértices, arestas e faces, respectivamente, de um polítopo regular, em $\mathbb{R}^3$. Então,*

$$V - A + F = 2$$

O Teorema 2.4.3 encontra-se demonstrado em [24].

Embora os poliedros e os polítopos sejam conjuntos que ficam completamente caracterizados pelo conjunto dos seus vértices ou pelas equações dos hiperplanos que limitam os semi-espaços que os formam, as descrições alternativas dos seus elementos revelam-se de grande importância.

Teorema 2.4.4 *Em $\mathbb{R}^n$, seja S um conjunto convexo fechado, com um número finito de faces. Então, S é um poliedro.*

Demonstração: Seja S um conjunto do espaço n-dimensional $\mathbb{R}^n$.
Como S é um conjunto convexo fechado, temos que S é a intersecção dos semi-espaços

fechados tangentes que o contém (ver Teorema 1.3.22).

Seja H um hiperplano que limita o semi-espaço tangente fechado que contém S. Da Definição 1.3.21 tem-se que existe um $x \in S$ tal que H é o único hiperplano suporte de S em x. O conjunto $F = S \cap H$ é uma face exposta e H é o único hiperplano suporte que contém F.

Sabemos que S tem, por hipótese, um número finito de faces. Logo, S admite um número finito de semi-espaços tangentes fechados, isto é, S é a intersecção de um número finito de semi-espaços fechados. Consequentemente, S é um poliedro.

$\diamondsuit$

Teorema 2.4.5 *Seja P um subconjunto de $\mathbb{R}^n$. As proposições seguintes são equivalentes:*

i) P é um polítopo;

ii) P é um poliedro limitado.

Demonstração: ($i \Leftarrow ii$) Seja P um poliedro limitado. Por definição de poliedro temos que P será fechado. Logo, P é um conjunto compacto. Então, pelo Teorema de Krein-Milman, $P = \overline{CH(ext\,(P))}$, onde $ext(P)$ designa o conjunto dos pontos extremos de P. Consequentemente, P será o invólucro convexo de um conjunto finito de pontos, ou seja, P será um polítopo.

$i \Rightarrow ii$) Seja P um polítopo, em $\mathbb{R}^n$. Se P é um polítopo, então ele será compacto e terá um número finito de faces. Assim, pelo Teorema 2.4.4, P será um poliedro limitado.

$\diamondsuit$

Num poliedro de dimensão n, as faces assumem diversas representações de acordo com a dimensão considerada. Assim, as faces de dimensão zero são os **vértices**, as faces cuja dimensão é um são as **arestas** e as faces de dimensão $n-1$ designam-se por **facetas**. Considerando, em $\mathbb{R}^3$, uma pirâmide quadrangular, conseguimos identificar cinco faces de dimensão zero, oito faces cuja dimensão é um e cinco faces de dimensão 2.

Descrever toda a estrutura facial de um poliedro consiste em descrever as suas faces qualquer que seja a sua dimensão, assim como, as relações de incidência entre elas. Considerando a definição de face e de semi-espaço, verificamos que cada face F de um conjunto C, em $\mathbb{R}^n$, é-nos dada pela sua intersecção com o semi-espaço que a contém.

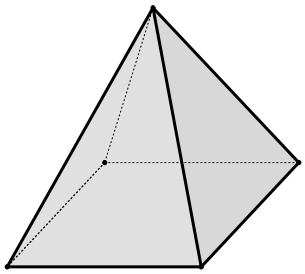


Figura 2.13: Pirâmide Quadrangular.

Observação 2.4.6 *Qualquer conjunto convexo C, em $\mathbb{R}^n$, possui como faces o próprio conjunto C (face de dimensão máxima) e o conjunto vazio (face de dimensão 0), $\emptyset$, os quais são designados por* ***faces impróprias****. Às restantes faces de C denominamos* ***faces próprias****.*

Teorema 2.4.7 *Em $\mathbb{R}^n$, todo o polítopo tem um número finito de faces, cada uma das quais é um polítopo.*

Demonstração: Seja um polítopo $P = CH(\{p_1, ..., p_m\})$, onde $\{p_1, ..., p_m\} \in \mathbb{R}^n$. Pelo Teorema de Krein-Milman sabemos que cada face F de P é o invólucro convexo dos seus pontos extremos. O Teorema 2.3.6 *ii*) permite-nos concluir que cada ponto extremo de F é também ponto extremo de P, logo F é o invólucro de algum subconjunto de vértices de P. Assim, desde que $\{p_1, ..., p_m\}$ contenha um vértice de P, F será o invólucro convexo de algum subconjunto de $\{p_1, ..., p_m\}$. $\diamondsuit$

Teorema 2.4.8 *O invólucro convexo da união de dois polítopos $P := CH\{x_1, ..., x_k\}$ e $Q := CH\{y_1, ..., y_r\}$, em $\mathbb{R}^n$, é um polítopo convexo: $CH(P \cup Q) = CH(CH\{x_1, ..., x_k\} \cup$ $\cup\, CH\{y_1, ..., y_k\}) = CH(x_1, ..., x_k, y_1, ..., y_r)$.*

A prova do Teorema 2.4.8 é evidente tendo em conta a definição de polítopo.

Capítulo 3

Algoritmos para a construção de invólucros convexos

No início deste capítulo, apresentamos alguns problemas práticos que podem ser modelados como problemas de construção do invólucro convexo de um conjunto finito de pontos de $\mathbb{R}^2$ ou $\mathbb{R}^3$. Em seguida, são apresentados algoritmos que permitem a construção do invólucro convexo de um dado conjunto finito de pontos no plano. Inicialmente, serão expostos dois algoritmos de força bruta (também conhecidos como algoritmos ingénuos) e, posteriormente, algoritmos mais eficientes. Para cada algoritmo apresentado será efectuada uma análise sob o ponto de vista da complexidade e, sempre que oportuno, far-se-á uma comparação com a complexidade de outros algoritmos. Finalmente, apresentam-se dois algoritmos que constroem invólucros convexos no espaço tridimensional.

3.1 Motivações

Actualmente, conhecem-se vários algoritmos que determinam invólucros convexos de conjuntos dados em tempo que se considera óptimo. No entanto, a Geometria Computacional continua a contar com instrumentos cada vez mais eficazes e com uma grande comunidade de pesquisadores, que fazem com que o estudo de problemas tais como o problema determinação de invólucros convexos desperte ainda muito interesse. A cons-

trução de invólucros convexos tem assumido um papel fundamental em diversas áreas das Ciências da Computação e em áreas tão diversas, como por exemplo, a Medicina e a Química.

Em Medicina, por exemplo, os invólucros convexos têm assumido particular importância no diagnóstico do Nódulo Pulmonar Solitário. Aqui, a geometria pode ajudar a distinguir um nódulo benigno de um nódulo maligno, uma vez que quando os nódulos assumem uma forma arredondada ou uma forma bem definida, provavelmente o nódulo será benigno. Ao contrário do que sucede quando os nódulos assumem formas mal definidas que traduzem, normalmente, um nódulo maligno. Este género de diagnóstico pode ser auxiliado por um algoritmo onde se recorre à construção 3D de invólucros convexos [44].

Em Química, os invólucros convexos podem ser utilizados para averiguar se uma alegada mistura com n componentes pode ser obtida recorrendo a misturas anteriores. Por exemplo, se M_1 for uma mistura constituída por 10 por cento de um componente X e 35 por cento de um componente Y, M_2 uma mistura constituída por 16 por cento do componente X e 20 por cento do componente Y e M_3 uma mistura com 13 por cento do componente X e 22 por cento do componente Y. Se representarmos M_1, M_2 e M_3 recorrendo aos pontos de coordenadas $p_1 = (0,10; 0,35)$, $p_2 = (0,16; 0,20)$ e $p_3 = = (0,07; 0,15)$, respectivamente, verificamos que qualquer ponto do triângulo definido pelos pontos p_1, p_2 e p_3 (invólucro convexo dos pontos p_1, p_2 e p_3) representa diferentes misturas. Assim, para sabermos se uma dada mistura que contém x por cento de X e y por cento de Y pode ser obtida a partir das misturas M_1, M_2 e M_3, basta averiguar se o ponto (x, y) que representa essa mistura, pertence ou não ao triângulo p_1, p_2, p_3.

O facto do problema da determinação do invólucro convexo de um conjunto de pontos ser um problema de simples resolução que, no pior dos casos, é um polítopo contendo o mesmo número de vértices do próprio conjunto e o facto deste problema ser muitas vezes utilizado noutros algoritmos sobre conjuntos de pontos, constituem motivações no estudo deste problema.

3.2 Complexidade Computacional

Tendo como principal objectivo a escolha de algoritmos mais eficientes e adequados a cada situação, torna-se imprescindível uma abordagem mais detalhada do conceito de complexidade de um algoritmo, para que se possa efectuar uma comparação entre os algoritmos que resolvem um mesmo problema.

3.2.1 Complexidade Algorítmica

Definição 3.2.1 *Um **algoritmo** é uma sequência ordenada e finita de instruções que levam à resolução de um dado problema.*

No nosso caso, iremos estudar algoritmos que nos permitem resolver o seguinte problema: *dado um conjunto finito de pontos do plano ou do espaço* $S = \{p_1, ..., p_n\}$, $n \geq 3$, *obter o invólucro convexo de* S.

Existem vários algoritmos que permitem construir o invólucro convexo de um conjunto finito de pontos. Neste trabalho, analisaremos cada algoritmo quanto à sua eficiência e para tal iremos determinar a sua complexidade. O conceito de eficiência de um algoritmo relaciona-se com o seu tempo de execução, isto é, o tempo que ele leva a produzir uma solução satisfatória para o nosso problema (ou a determinar que tal solução não pode ser obtida) e o espaço de memória (em computador) necessário para produzir esta mesma solução. Tendo em conta o tempo que um algoritmo demora a produzir a solução de um problema e o espaço de memória utilizado podemos distinguir dois tipos de complexidade: a *complexidade temporal* e a *complexidade espacial*. Assim, quando escolhemos um algoritmo devemos optar por aquele que processa a solução em menos tempo e que utiliza o menor espaço na organização dos dados do problema.
O estudo da eficiência de cada algoritmo tem por base a variação do tempo de resolução do problema em função do tamanho do conjunto de entrada [1]. Assim, se A for um algoritmo que permite a resolução de um problema em m passos, dizemos que A corre em tempo m.

[1] Aqui, o tamanho significa o número de elementos do conjunto.

No que diz respeito à análise de um algoritmo podemos distinguir dois tipos: a análise no *caso esperado* (ou *caso médio*) e no *pior caso*. A análise para o caso esperado, corresponde ao ideal pois dá-nos uma informação mais precisa acerca da complexidade algorítmica. No entanto, ela é pouco usual uma vez que é necessário que os dados do problema surjam seguindo um modelo de distribuição probabilístico, o que na práctica pode ser difícil de encontrar. Relativamente à análise do pior caso, ela pode não ser tão elucidativa da realidade, pois o pior caso pode corresponder a um caso particular ou ainda a um caso raro. Contudo, daremos maior ênfase àquela em que os algoritmos têm como entrada um conjunto de tamanho infinito, isto é, a análise assimptótica que corresponde ao pior dos casos.

Sejam A e B dois algoritmos que resolvem um mesmo problema e $T(n)$ o desempenho do algoritmo numa instância de tamanho n. Para escolhermos qual dos algoritmos possui um melhor desempenho temos de efectuar uma análise assimptótica de $T_A(n)$ e $T_B(n)$, ou seja, teremos de comparar $T_A(n)$ com $T_B(n)$ quando n cresce. Caso se venha a verificar que $T_A(n)$ cresce mais rápido, então podemos dizer que o algoritmo B terá uma maior eficiência, uma vez que permitirá resolver problemas com maiores dimensões em menos tempo.

Para podermos estudar melhor, sob o ponto de vista da eficiência, um algoritmo que nos permita resolver o problema da construção do invólucro convexo teremos como referência um modelo computacional que nos irá permitir inferir propriedades matemáticas dos algoritmos. O modelo escolhido será o das **árvores de decisões algébricas**. Segundo este modelo, os algoritmos procedem a uma avaliação sequencial das expressões algébricas considerando o conjunto de valores de entrada e tomando decisões consoante o sinal desses valores. Mais precisamente, como entrada teremos um conjunto finito de números reais $x_1, x_2, x_3, ..., x_n \in \mathbb{R}$, correspondentes às coordenadas e equações que representam certos objectos geométricos, e um algoritmo representado por uma árvore binária cujas folhas possuem as soluções possíveis do problema.

Exemplo 3.2.2 *Vejamos uma aplicação deste modelo computacional na resolução do problema da ordenação de três números reais x_1, x_2 e x_3. Como veremos mais adiante, a resolução deste problema será muito útil e auxiliará alguns dos algoritmos que iremos estudar.*

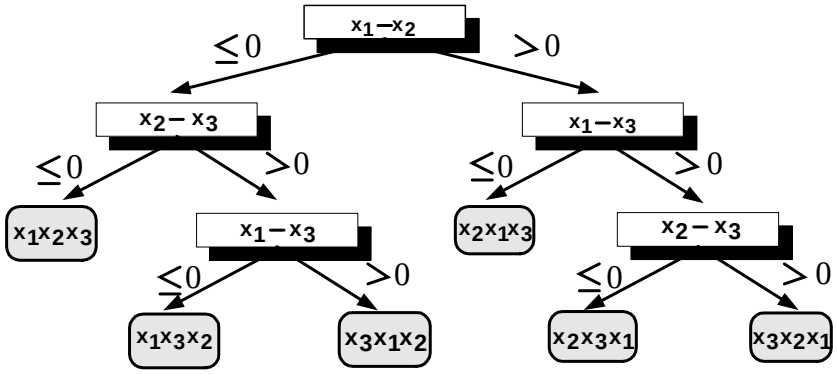


Figura 3.1: Ordenação de três números numa àrvore de decisões algébricas.

Temos, assim, um algoritmo cuja representação recorre a uma árvore binária, onde cada folha, correspondente a um nó terminal, dá-nos uma possível resposta ao problema inicial. Por outro lado, cada nó interno corresponde a uma instrução do algoritmo, na qual é avaliada a sequência inicial de pontos. Considerando o sinal do resultado de cada operação, o algoritmo segue para o nó da direita ou para o nó da esquerda. Note-se que no final teremos um total de seis folhas contendo, cada uma delas, uma das permutações possíveis dos três números.

A complexidade de um algoritmo representado através de uma árvore de decisões algébricas é medida pelo número de nós percorridos até à solução do problema. Neste modelo computacional, todas as operações elementares terão o mesmo custo e a complexidade do algoritmo ficará definida pelo número máximo de nós percorridos, o que corresponde ao pior caso.

Definição 3.2.3 *Sejam P um problema, A um algoritmo que resolve P e I uma instância de P. Denota-se por $T_A(I)$ e chama-se desempenho do algoritmo A, ao custo de A para resolver a instância I de P, ou seja, o número de nós da árvore que representa A que são percorridos até a solução de P com a entrada I.*

Suponhamos que existem dois algoritmos que resolvem um mesmo problema. Perante esta situação e dado que se pretende escolher aquele que é mais eficiente, teremos de optar por aquele que possui a instância mais desfavorável, ou seja, o que corresponde ao pior caso. Mais precisamente, para instâncias de tamanho n teremos:

$$T_A(n) = \max_{|I|=n} T_A(I).$$

Ao representarmos algoritmos através de uma árvore binária finita, a solução só é permitida para instâncias que têm o mesmo tamanho. Por exemplo, se considerarmos o exemplo da ordenação de três números, o algoritmo representado na árvore binária só irá conseguir ordenar três números. Assim, para resolver problemas de instâncias de maior dimensão usa-se o mesmo modelo só que desta vez a árvore binária ficará constituída por uma família de árvores, todas elas finitas e cada uma com o tamanho de instância. Para vermos como é que essas árvores crescem em função do tamanho n das instâncias basta-nos medir a profundidade da árvore, isto é, averiguar quantos nós é que são percorridos desde a raiz até a uma folha.

O conjunto de notações seguintes são usadas na análise da complexidade de um algoritmo. Estas notações descrevem o tempo de execução de um algoritmo e encontram-se definidas em termos de funções que têm como domínio o conjunto $\mathbb{N} = \{1, 2, 3, ...\}$.
Seja $S(n)$ a quantidade de memória exigida por um algoritmo em função do tamanho do problema (n) e seja $T_A(n)$ o tempo de execução de um algoritmo A em função do tamanho (n) do problema.

Nas notações que se seguem f e g designam funções de domínio $\mathbb{N}$.

A notação seguinte designa a *ordem de complexidade* de um algoritmo e descreve o seu tempo de execução para o pior caso. Encontra-se definida da seguinte forma:

- $O(g(n))$: O conjunto de todas as funções $f(n)$ tais que $\exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} : |f(n)| \leq c|g(n)|, \ \forall n \geq n_0$, isto é, $f(n) = O(g(n))$ quando existem $c \in \mathbb{R}^+$ e $n_0 \in \mathbb{N}$ tais que $|f(n)| \leq c|g(n)|, \forall n \geq n_0$.

Esta notação é usada quando pretendemos o limite superior assimptótico. Neste sentido, a classe de funções $O(g(n))$ limitam superiormente o tempo de execução de um algoritmo, qualquer que seja o seu conjunto de entrada.

Definição 3.2.4 *A complexidade temporal de um algoritmo A é* $O(g(n))$ *se existir uma constante* $c \in \mathbb{R}^+$ *desde que para todo o inteiro* $n \geq 0$*, A correr no mínimo em* $c\, g(n)$ *qualquer que seja a entrada de tamanho n.*

- $\Omega(g(n))$: O conjunto de todas as funções $f(n)$ tais que $\exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} : c|g(n)| \leq |f(n)|,\ \forall n \geq n_0$, isto é, dizemos que $f(n) = \Omega(g(n))$ quando existem $c \in \mathbb{R}^+$ e $n_0 \in \mathbb{N}$ tais que $c|g(n)| \leq |f(n)|$.

Neste caso, $\Omega(g(n))$ é a notação usada sempre que se pretende um limite inferior assimptótico, ou seja, a classe de funções que limitam inferiormente o tempo de execução do algoritmo, qualquer que seja o seu conjunto de entrada.

- $\Theta(g(n))$: O conjunto de todas as funções $f(n)$ tais que $\exists c_1, c_2 \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} : c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|,\ \forall n \geq n_0$, isto é, dizemos que $f(n) = \Theta(g(n))$ quando $f(n)$ é $O(g(n))$ e $f(n)$ é $\Omega(g(n))$, ou seja, $c_1\ g(n) \leq f(n) \leq c_2\ g(n), \forall c_1, c_2 \in \in \mathbb{R}, \forall n \geq n_0$.

Neste sentido, a função $f(n) = \Theta(g(n))$ sempre que existirem duas constantes positivas tais que sendo multiplicadas pelo módulo de $g(n)$ elas limitam o módulo de f.

Notemos que, quando na descrição das definições anteriores escrevemos que $f(n) = = O(g(n))$, $f(n) = \Omega(g(n))$ e $f(n) = \Theta(g(n))$, a relação "=" é assimétrica. O que significa que quando escrevemos, por exemplo, $f(n) = O(g(n))$, pretendemos dizer que $f(n)$ pertence a $O(g(n))$.

Os gráficos da Figura 3.2 ajudam-nos a compreender melhor as notações anteriores:

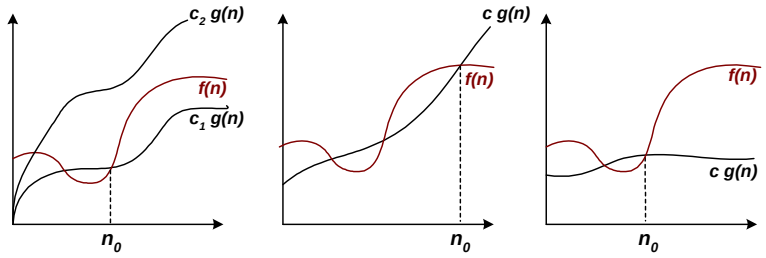


Figura 3.2: Ilustração das notações $\Theta(g(n))$, $O(g(n))$ e $\Omega(g(n))$, respectivamente.

As funções seguintes são exemplos de funções onde podemos visualizar um crescimento assimptótico. Estas funções encontram-se ordenadas por ordem estritamente crescente, o que faz com que se $g(n)$ vier depois de $f(n)$, então podemos dizer que $f(n) = O(g(n))$:

$$log\, n < n^{\frac{1}{k}} < \sqrt{n} < n < n\, log\, n < n^2 < n^k < 2^n < n! < n^n, \tag{3.2.1}$$

para $n \geq 4, k \in \mathbb{N}, k > 1$.
No entanto, $f(n) \neq \Theta(g(n))$.

Partindo da notação O é possível distinguir classes de complexidade algorítmica, como podemos observar nos exemplos da Tabela 3.1:

$O(1)$	Constante
$O(\log\ n)$	Logarítmica
$O(n)$	Linear
$O(n\ \log\ n)$	Logarítmica
$O(n^2)$	Quadrática
$O(n^3)$	Cúbica
$O(n^k)$	Polinomial
$O(a^n)$	Exponencial
$O(n!)$	Factorial

Tabela 3.1: Exemplos de classes de complexidade algorítmica.

3.2.2 Complexidade de Problemas

Vejamos o que sucede quando em vez de avaliarmos a complexidade de algoritmos se avalia a complexidade de problemas.

Definição 3.2.5 *Seja P um problema e A um algoritmo que resolve P. Define-se complexidade do problema P e denota-se por $T_P(n)$ como sendo o melhor que um algoritmo A consegue fazer, isto é, $T_P(n) = \min_A T_A(n)$, onde $T_A(n)$ designa a complexidade do algoritmo A para resolver a instância n de P.*

Definição 3.2.6 *Seja A um algoritmo que resolve P. Dizemos que A é óptimo quando $T_A = \Theta(T_P)$ ou, equivalentemente, $T_P = \Theta(T_A)$.*

Estimar a complexidade de um problema pode tornar-se num processo complicado. Como já foi referido anteriormente, encontrar uma estimativa do tipo $O(g)$, ou seja, do limite superior é mais simples, pois corresponde a encontrar um qualquer algoritmo que resolva o problema, qualquer que seja a sua complexidade. Contudo, a busca de um limite inferior, ou seja, de uma estimativa do tipo $\Omega(g)$ pode tornar-se num processo complicado embora, em certos casos, esse limite inferior seja trivialmente encontrado. Por exemplo, se considerarmos um problema onde se pretende ordenar n pontos, o seu limite inferior será $\Omega(n)$, pois intuitivamente serão testados n pontos.
Nos algoritmos apresentados no decorrer das secções seguintes os limites inferiores não serão abordados dado que se desenquadram do objectivo central deste trabalho.

Muitos dos algoritmos para a determinação do invólucro convexo de um conjunto de pontos iniciam-se com a ordenação dos pontos que constituem o seu conjunto de entrada. Assim sendo, torna-se útil avaliarmos primeiro a complexidade do seguinte problema:

Ordenação de n pontos: Dados um conjunto de n pontos no plano $\{p_1, p_2, ..., p_n\}$, encontrar uma permutação Π do conjunto de índices $\{1, 2, 3, ..., n\}$ tal que $p_{\Pi(1)} \leq p_{\Pi(2)} \leq \ldots \leq p_{\Pi(n)}$.

Teorema 3.2.7 *No modelo computacional das árvores de decisões algébricas, o algoritmo para a ordenação de n pontos tem complexidade temporal* $\Omega(n \log n)$.

Demonstração: Seja A um algoritmo que ordena n pontos. Usando o modelo computacional descrito no Exemplo 3.2.2, A poderá ser representado recorrendo a uma árvore binária. Se o objectivo é ordenar n pontos, na solução final teremos pelo menos $n!$ folhas, correspondentes às soluções possíveis para o nosso problema de ordenação. Por outro lado, uma árvore binária de profundidade p tem, no máximo, 2^p folhas. Deste modo, $2^p \geq n!$. Logo, $T_A(n) = p = \log_2(2^p) \geq \log_2 n!$ e, portanto, $T_A(n) = \Omega(\log_2 n!)$. Considerando $n \geq 4$ verificamos que $n! \geq 2^n$. Logo, $T_A(n) \geq \log_2 n! \geq \log_2 2^n = n$. O que mostra o resultado que foi trivialmente considerado como limite inferior para a ordenação de n pontos, ou seja, que $T_A(n) = \Omega(n)$.
Encontre-se, agora, um limite inferior não trivial. Para tal, estime-se $\log_2 n!$ comparando $n!$ com n^n. De (3.2.1) temos que $n! \leq n^n$. Agora, para encontrar um limite inferior não trivial estimaremos $(n!)^2$ em comparação com n^n.

$(n!)^2 = (1.2.3...(n-2).(n-1).n).(n.(n-1).(n-2)...3.2.1)$
$(n!)^2 = ((1.n).(2.(n-1)).(3.(n-2))...((n-2).3).((n-1).2).(n-1) \geq n^n$

Podemos, então, concluir que $(n!)^2 \geq n^n$. Donde, $\log_2 n! \geq \frac{1}{2}\log_2(n^n) = \frac{1}{2} n \log_2 n$ e, consequentemente, $T_A(n) = \Omega(\log n!) = \Omega(n \log n)$.

$\diamondsuit$

A ordenação de n pontos pode ser feita, sob o ponto de vista algorítmico, recorrendo a diferentes processos tais como: força bruta, selecção, inserção e intercalação. No entanto, estes processos não apresentam a mesma capacidade de resposta a este problema, apresentando complexidades muito diferentes. Ao recorrermos, por exemplo, a um algoritmo de força bruta para ordenar n pontos $\{p_1, p_2, ..., p_n\}$ teremos de considerar as $n!$ permutações Π do conjunto de índices $\{1, 2, 3, ..., n\}$ e verificar se $p_{\Pi(1)} \leq p_{\Pi(2)} \leq ... \leq p_{\Pi(n)}$. Sabemos que uma dessas permutações irá corresponder

à solução pretendida. Contudo, sob o ponto de vista de complexidade, este tipo de algoritmos despertam pouco interesse uma vez que o seu desempenho é lento. Como se verá mais adiante, um algoritmo de força bruta tem complexidade superior a $O(n^2)$.
No que se refere aos processos por selecção e por inserção, estes apresentam complexidade quadrática. No processo por selecção, os pontos encontram-se armazenados numa lista e depois são escolhidos por ordem crescente. Nesta lista, o ponto $p_{\Pi(k)}$ corresponde ao ponto que se encontra na posição k. Depois, para efectuar a selecção terá de ser identificado o menor dos $n-k+1$ pontos que ainda não foram seleccionados, isto é, terão de ser feitas $n-k$ comparações:

$$(n-1)+(n-2)+(n-3)+...+2=\tfrac{n(n-1)}{2}-1.$$

Temos, portanto, uma complexidade quadrática que é inferior à do algoritmo de força bruta.
No processo por inserção os pontos são ordenados incrementalmente, ou seja, quando chegamos ao passo k já se encontram ordenados os pontos $\{p_1, p_2, ..., p_k\}$ e no final para o passo de ordem n teremos todos os pontos ordenados. Logo, o número máximo de comparações será:

$$1+2+...+(n-1)=\tfrac{n(n-1)}{2}\ ,$$

o que também corresponde a uma complexidade quadrática.

Finalmente, na ordenação por intercalação encontra-se um algoritmo muito mais eficiente o qual é conhecido por **Algoritmo de ordenação Mergesort** [24]. Este algoritmo baseia-se no paradigma da divisão e conquista, isto é, ele obtém a solução pretendida combinando soluções dos seus sub-problemas.
Os algoritmos por divisão e conquista assentam em três etapas fundamentais: a *divisão*, a *recursão* e a *combinação*. Na primeira etapa, efectua-se a divisão do conjunto em dois subconjuntos, na segunda aplica-se recursivamente o algoritmo a cada subconjunto e na terceira etapa, para obter a solução do problema original, combinam-se os resultados obtidos na etapa anterior.

Suponhamos que queremos ordenar o conjunto S de n pontos recorrendo a um algoritmo por intercalação. Para tal devemos começar por dividir S em dois subconjuntos com a mesma dimensão (ou dimensão aproximada), ordenando os pontos que se encontram em cada um deles. Em seguida, aplicamos recursivamente o mesmo algoritmo a cada subconjunto.

Vejamos o que sucede em termos de complexidade $T(n)$. De salientar que, tendo as duas primeiras etapas complexidade proporcional a n, $T(n)$ irá satisfazer:

$$T(n) = T(\lfloor \tfrac{n}{2} \rfloor) + T(\lceil \tfrac{n}{2} \rceil) + cn, \text{ onde } c \text{ é uma constante.}$$

Então, temos que $T(n) = 2T(\frac{n}{2}) + cn$ e $T(\frac{n}{2}) = 2T(\frac{n}{4}) + c(\frac{n}{2})$. Logo, $T(n) = 4T(\frac{n}{4}) + 2cn$. Para simplificar os cálculos, considere-se n como uma potência de base dois, ou seja, $n = 2^k$. Obtemos assim,

$$T(n) = 2^k T(\tfrac{n}{2^k}) + k\, c\, n = nT(1) + c\, n \log_2 n = O(n\, log\, n).$$

De facto, este algoritmo mostra-se eficiente conduzindo-nos a uma complexidade óptima.

3.3 Algoritmos para a construção de invólucros convexos no plano

Nesta secção, para facilitar a apresentação de cada algoritmo considera-se que os pontos encontram-se em posição geral, isto é, parte-se do pressuposto de que não existem três pontos colineares.

3.3.1 Formulação do problema e conceitos básicos

Dado um conjunto S de n pontos, em $\mathbb{R}^2$, encontrar um algoritmo que permita a construção do seu invólucro convexo corresponde a determinar quais os pontos de S que são vértices de $CH(S)$, ordenando-os de acordo com a sua ocorrência na fronteira do $CH(S)$.

Definição 3.3.1 *Sejam p_i, p_j e p_k três pontos, não colineares, de um dado conjunto, em $\mathbb{R}^2$. Diz-se que os pontos p_i, p_j, p_k fazem* ***curva à direita*** *se o produto escalar entre dois vectores consecutivos $\overrightarrow{p_ip_j}$ e $\overrightarrow{p_jp_k}$ for positivo, ou seja, $\langle p_j - p_i, p_k - p_j \rangle > 0$. Se, pelo contrário, o produto externo entre $\overrightarrow{p_ip_j}$ e $\overrightarrow{p_jp_k}$ for negativo, ou seja, $\langle p_j - p_i, p_k - p_j \rangle < 0$, diz-se que os pontos p_i, p_j e p_k fazem uma* ***curva à esquerda****.*

Definição 3.3.2 *Sejam p_i, p_j e p_k três pontos do plano, não colineares, que definem um triângulo. Diz-se que o triângulo $\Delta[p_ip_jp_k]$ tem* ***orientação positiva*** *se os pontos p_i, p_j, p_k fizerem uma curva à esquerda em cada ponto extremo. Se, pelo contrário, os pontos p_i, p_j, p_k fizerem uma curva à direita diz-se que o triângulo $\Delta[p_ip_jp_k]$ tem* ***orientação negativa*** (ver Figura 3.3).

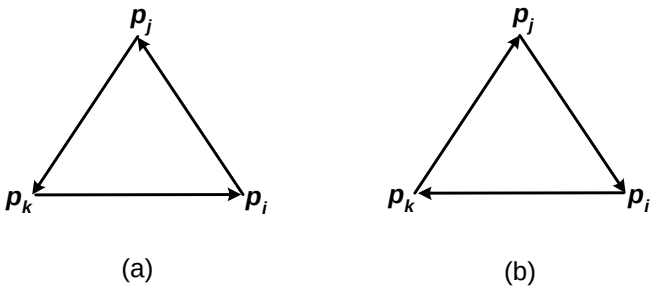


Figura 3.3: (a) Orientação positiva ; (b) Orientação negativa.

A verificação de que um dado ponto p_k, se encontra ou não à esquerda de uma recta orientada p_ip_j (recta cujo vector director é $\overrightarrow{p_ip_j}$) equivale a verificar se o triângulo definido pelo terno de pontos (p_i, p_j, p_k) tem orientação positiva. Esta verificação é, por várias vezes, utilizada nos algoritmos que estudaremos em seguida. Neste sentido, serão utilizados dois predicados que utilizam esta verificação: o predicado *Left* e o predicado *LeftOn*. O predicado *Left* recebe como parâmetros três pontos (p_i, p_j, p_k), determina se um dado ponto p_k está à esquerda ou à direita da recta orientada definida pelos pontos p_i e p_j, devolvendo verdadeiro apenas se o ponto p_k se encontrar à esquerda da recta. O predicado *LeftOn* difere do predicado *Left* uma vez que devolve verdadeiro também quando o ponto em análise se encontra sobre a recta orientada p_ip_j.

Definição 3.3.3 *Um conjunto de pontos P, em $\mathbb{R}^2$, encontra-se* ***ordenado por ângulo polar*** *relativamente a um ponto $p \in P$, se a partir de um ponto p se traça um raio r com origem em p que efectua um varrimento em sentido positivo (anti-horário) sobre os pontos do conjunto P e à medida que o raio r vai intersectando os pontos, o ângulo polar definido pelo raio r e os pontos do conjunto aumenta monotonamente.*

Definição 3.3.4 *Um conjunto de pontos P, em $\mathbb{R}^2$, encontra-se* ***ordenado lexicograficamente****, se após os pontos de P estarem ordenados por ângulo polar é considerado o ponto com maior distância à origem, eliminando-se o que está mais próximo da origem.*

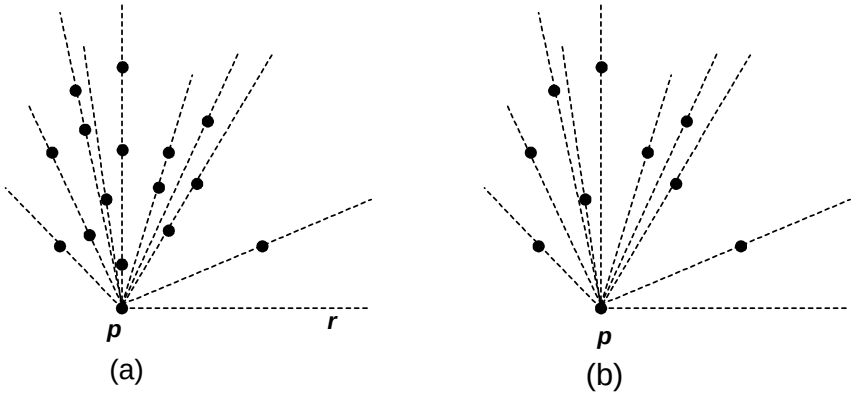


Figura 3.4: (a) Ordenação por ângulo polar ; (b) Ordenação lexicográfica.

Definição 3.3.5 *Um* ***vértice*** *de um polígono simples diz-se* ***convexo*** *se a amplitude do ângulo interno, formado pelas arestas nele incidentes, for menor ou igual a π. Caso contrário, o vértice diz-se* ***reflexo*** *ou* ***côncavo****.*

Observação 3.3.6 *Prova-se facilmente que cada polígono tem pelo menos um vértice convexo e que todos os vértices convexos de um polígono são vértices do seu invólucro convexo.*

Definição 3.3.7 *Uma* ***cadeia poligonal simples*** *diz-se* ***convexa*** *se todos os ângulos interiores ao polígono simples limitado por esta cadeia forem convexos.*

Definição 3.3.8 *Uma **cadeia poligonal** diz-se **monótona** relativamente a uma recta l, se qualquer recta ortogonal a l e que intersecte a cadeia poligonal, a intersecta somente num vértice ou numa só aresta* (ver Figura 3.5).

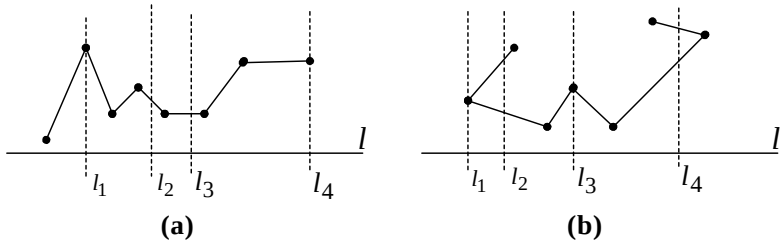


Figura 3.5: (a) Cadeia poligonal monótona em relação à recta l ; (b) Cadeia poligonal não monótona em relação à recta l.

Definição 3.3.9 *Seja P um polígono simples. Diz-se que P é um **polígono estrelado** se existe pelo menos um ponto $x \in P$ tal que para todo o ponto $y \in P$ o segmento $[x, y] \subset P$* (ver Figura 3.6).

Os polígonos estrelados possuem no seu interior pelo menos um ponto p com a particularidade de conseguir *"ver"* cada um dos restantes pontos do polígono.

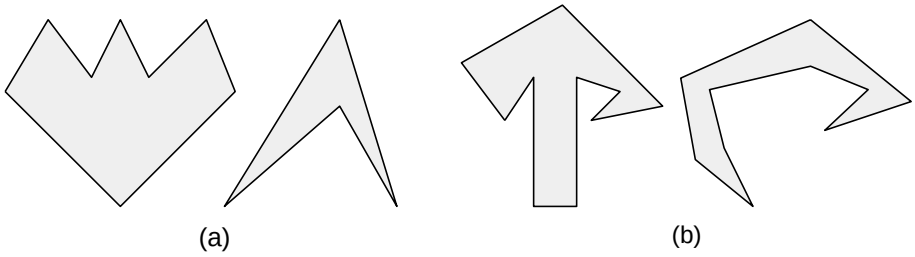


Figura 3.6: (a) Polígonos estrelados ; (b) Polígonos não estrelados.

3.3.2 Algoritmos de Força Bruta

Os dois algoritmos apresentados, em seguida, resolvem o problema da construção do invólucro convexo, em $\mathbb{R}^2$, tendo por base a definição de ponto extremo (ou *vértice*) e de

aresta extrema (ou *aresta suporte*). Estes algoritmos são designados, respectivamente, por **Algoritmo dos Pontos Não Extremos** e **Algoritmo das Arestas Extremas**.

Sendo S um conjunto convexo com n pontos, o algoritmo que se segue tem por base a Definição 2.1.10 *vii*) e o Lema 2.3.2. Para construir o $CH(S)$, o Algoritmo dos Pontos Não Extremos identifica os pontos que se encontram no interior de um dado triângulo, ou seja, os pontos não extremos.

No **Algoritmo dos Pontos Não Extremos** podemos identificar duas fases fundamentais:

1. Identificação dos pontos não extremos de S;

2. Ordenação dos pontos extremos de forma a que eles formem um conjunto convexo o que, em $\mathbb{R}^2$, equivale a encontrar um polígono convexo.

Na primeira fase, a verificação de que um ponto se encontra ou não no interior de um triângulo definido por três pontos de S é feita recorrendo à primitiva *Left*, ou seja, percorrendo o triângulo no sentido positivo testa-se se o ponto se encontra à esquerda das três arestas que definem o triângulo. Se tal acontecer o ponto estará no interior do triâgulo e, consequentemente, não será ponto extremo.

Pseudo-código do Algoritmo dos Pontos Não Extremos (CH1)

Entrada: Um conjunto finito de pontos no plano, $S = \{p_1, p_2, ..., p_n\}$.
Saída: Pontos não extremos do conjunto S.

1. Para cada i fazer
2. Para cada $j, j \neq i$ fazer
3. Para cada $k, k \neq i, k \neq j$ fazer
4. Para cada $l, l \neq i, l \neq j, l \neq k$ fazer
5. Se p_l se encontra à esquerda ou sobre (p_i, p_j) e

6. p_l se encontra à esquerda ou sobre (p_j, p_k) e

7. p_l se encontra à esquerda ou sobre (p_k, p_i)

8. Então p_l é um ponto não extremo

Teorema 3.3.10 *O Algoritmo dos Pontos Não Extremos tem complexidade temporal* $O(n^4)$.

Demonstração: Para os n pontos do conjunto S terão de ser construídos um total de $C_3^{n-1} = (\frac{n-1}{3}) = O(n^3)$ triângulos, para verificar se p é ou não ponto interior de um triângulo. A fase correspondente à construção dos triângulos pode ser efectuada num número constante de operações e a verificação de que o ponto é ou não ponto extremo com complexidade temporal $O(n^3)$. Contudo, este procedimento terá de ser repetido para os n pontos do conjunto o que leva a uma complexidade total de $O(n^4)$. $\diamondsuit$

Este algoritmo de execução aparentemente simples é, no entanto, pouco eficiente uma vez que a sua complexidade é elevada.

Em seguida, analisa-se um algoritmo que identifica as arestas extremas de um conjunto convexo, isto é, as arestas do invólucro convexo. É, por isso, muito semelhante ao algoritmo apresentado anteriormente.

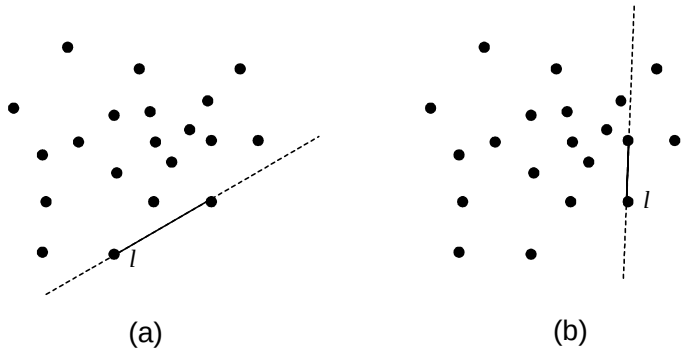


Figura 3.7: (a) Aresta extrema ; (b) Aresta não extrema.

Definição 3.3.11 *Seja S um conjunto de pontos no plano e l uma aresta definida por dois dos seus pontos. Diz-se que l é uma* ***aresta extrema*** *se os restantes pontos de S se encontrarem sobre l ou num dos semi-planos fechados por ela determinados* (ver Figura 3.7).

O Algoritmo Arestas Extremas inicia-se com a identificação das arestas extremas do invólucro. Uma vez encontradas estas arestas, podemos iniciar a construção da fronteira do invólucro convexo.

Pseudo-código do Algoritmo das Arestas Extremas (CH2)

Entrada: Um conjunto finito de pontos no plano, $S = \{p_1, p_2, ..., p_n\}$.
Saída: Arestas extremas do conjunto S.
1. Para cada i fazer
2. Para cada $j \neq i$ fazer
3. Para cada $k, k \neq i, k \neq j$ fazer
4. Se p_k se encontra à esquerda ou sobre (p_i, p_j)
5. Então (p_i, p_j) é aresta extrema.

Teorema 3.3.12 *O Algorimo das Arestas Extremas tem complexidade temporal $O(n^3)$.*

Demonstração: Em termos de complexidade temos de considerar um total de C_2^n arestas e verificar qual a posição dos n pontos do conjunto relativamente a cada aresta, logo teremos uma complexidade de $n\,(C_2^n) = O(n^3)$. ♢

Este algoritmo é mais eficiente do que o anterior é, no entanto, muito pouco eficiente.

3.3.3 Algoritmo de Jarvis

O algoritmo proposto por Jarvis [29], em 1972, surgiu pela primeira vez com Chand & Kapur [12] como um algoritmo que permitia a construção de invólucros convexos

qualquer que fosse a dimensão do espaço onde os pontos se encontravam. Durante vários anos foi considerado o primeiro algoritmo a permitir a construção de invólucros em espaços com mais de duas dimensões. Mais tarde, veio a verificar-se que este não funcionava em espaços de dimensão superior a dois.

O Algoritmo de Jarvis baseia-se no algoritmo das arestas extremas e simula o enrolar de um cordel em torno do conjunto de pontos S, pelo que também é conhecido como Algoritmo Embrulho-para-presente (ou *Gift-Wrapping*), construindo o $CH(S)$ de forma incremental ao procurar uma aresta de cada vez. Este algoritmo inicia-se com a determinação daquela que será a primeira aresta de $CH(S)$ e, em seguida, analisa os restantes pontos de S procurando, em sentido anti-horário, a próxima aresta e assim sucessivamente até que esteja formado um polígono convexo. Podemos dizer que a ideia geral deste algoritmo fica evidente no Teorema 3.3.13.

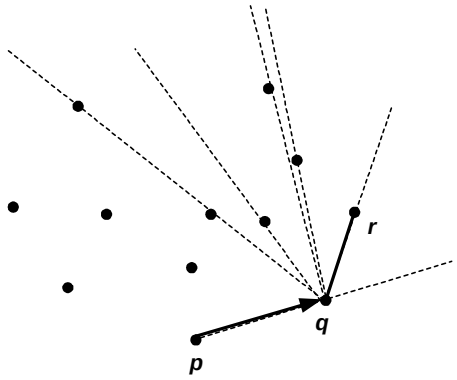


Figura 3.8: Identificação da segunda aresta de $CH(S)$.

Teorema 3.3.13 *Seja S um conjunto finito de pontos no plano e $CH(S)$ o seu invólucro convexo. Se $[p, q]$ é uma das arestas do $CH(S)$ orientada positivamente, então a próxima aresta será $[q, r]$, onde r é o ponto de S tal que $[q, r]$ define o menor ângulo com a semi-recta $[p, q]$* (ver Figura 3.8).

O algoritmo começa com a escolha do ponto de ordenada mínima (em caso de empate escolhe-se o ponto com maior abcissa). Neste caso, o ponto que tem ordenada mínima é o ponto p_1, o qual considera-se como ponto *pivot* (ver Figura 3.9). Note-se que

este ponto encontra-se garantidamente no invólucro convexo de S. O passo seguinte consiste em passar por p_1 uma recta suporte horizontal L, dirigida da esquerda para a direita, até que esta intersecte S num ponto. Perante esta situação é possível afirmar que não se encontram quaisquer pontos de S sobre L ou à sua direita. Em seguida, ordenam-se todos os pontos de S por ângulo polar e distância ao ponto p_1, tal como se pode observar na Figura 3.9.

Feita a ordenação dos pontos, o algoritmo segue com a rotação de L sobre p_1 em sentido positivo, até que L intersecte o próximo ponto do conjunto S, seja p_2. Desta forma é encontrada a primeira aresta do $CH(S)$ que corresponde ao segmento $[p_1, p_2]$, sendo p_1 e p_2 os vértices (ou vértices suporte).

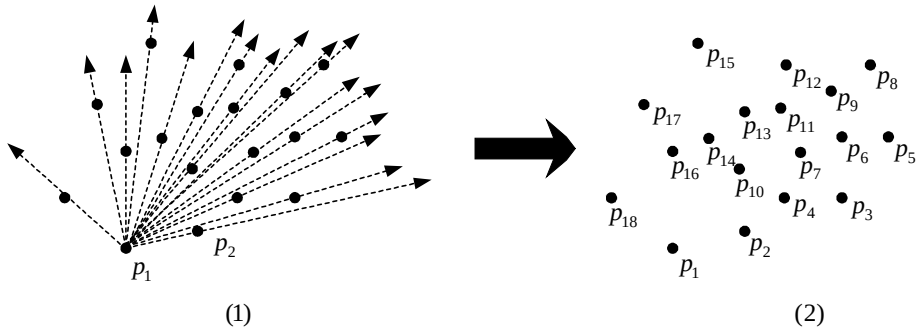


Figura 3.9: Algoritmo de Jarvis: ordenação do conjunto S.

Agora, p_2 irá funcionar como o novo ponto *pivot* e sobre ele será rotacionada a recta L até que esta intersecte um novo ponto de S, ponto p_3. O segmento $[p_2, p_3]$ irá pertencer à fronteira do $CH(S)$ e, portanto, constituirá uma nova aresta do invólucro. Para que o $CH(S)$ fique construído basta repetir o procedimento anteriormente descrito até que aquela que é considerada a última aresta do invólucro volte ao ponto de partida, ou seja, p_1 (ver Figura 3.10).

Por vezes, acontece que ao se efectuar a rotação da recta L sobre um ponto, ela intersecta, simultaneamente, dois ou mais pontos. Perante esta situação deve ser escolhido o ponto cuja distância é máxima para o ponto *pivot*, uma vez que se tal não acontecer existem pontos à direita da recta e ela deixa de ser uma aresta extrema.

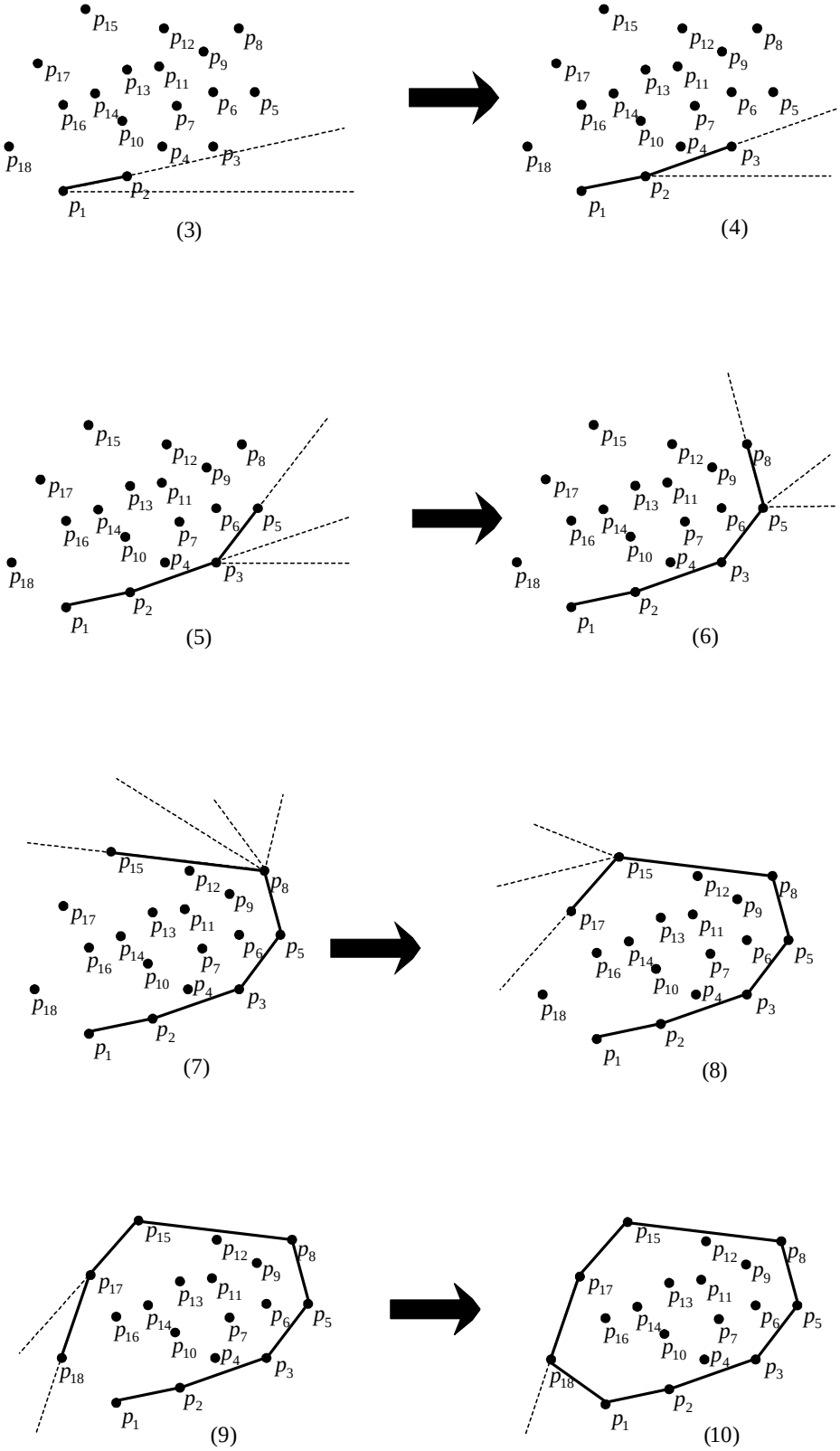


Figura 3.10: Algoritmo de Jarvis em acção.

Pseudo-código do Algoritmo de Jarvis (CH3)

Entrada: Um conjunto finito de pontos no plano, $S = \{p_1, ..., p_n\}$.

Saída: As arestas extremas de S à medida que vão surgindo, em sentido anti-horário, na fronteira do $CH(S)$.

1. Encontrar o ponto de ordenada mínima
2. Seja i_0 o índice desse ponto $i \leftarrow i_0$
3. Seja L a recta horizontal que passa por p_{i_0}
4. Repetir
5. Para cada $j, j \neq i$ fazer
6. Calcular o ângulo polar Θ_j do ponto p_j relativamente ao ponto p_i e à recta L
7. Seja k o índice do ponto com menor ângulo polar
8. Saída $[p_i, p_k]$ como aresta do $CH(S)$
9. $i \leftarrow k$
10. $L \leftarrow$ recta (p_i, p_k)
11. Até $i = i_0$

Analisando este algoritmo quanto à sua complexidade, podemos dizer que apresenta uma complexidade inferior à dos algoritmos apresentados anteriormente.

Teorema 3.3.14 *O Algoritmo de Jarvis tem complexidade temporal $O(hn)$, onde h é o número de arestas do invólucro convexo.*

Demonstração: O Algoritmo de Jarvis testa os $O(n)$ pontos do conjunto ao tentar encontrar a próxima aresta do invólucro, e a construção do $CH(S)$ tem no máximo h arestas, o que permite concluir que a complexidade temporal é de $O(n^2)$. Mas, ao efectuarmos uma análise mais cuidada da complexidade deste algoritmo verificamos que esta pode não ser tão elevada, dependendo directamente da instância considerada.
Seja h o número de arestas do $CH(S)$. No pior dos casos, correspondente à situação em que todos os pontos de S são pontos extremos o que acontece, por exemplo, quando

estes se encontram dispostos sobre um círculo, teremos que $h = O(n)$.
Outro caso particular surge quando se tem pontos colineares no conjunto e se tem $h = 2$, ou quando os pontos se encontram no interior de um triângulo e se tem $h = 3$. Neste caso, o algoritmo leva tempo $O(n)$ para encontrar as h arestas, o que perfaz uma complexidade linear $O(nh)$. ♢

3.3.4 Algoritmo de Graham

Segundo O'Rourke [24], a primeira publicação dentro da Geometria Computacional terá sido sobre o Algoritmo de Graham [26], em 1972, para a construção de invólucros convexos de um conjunto de pontos no plano. Nessa altura, este algoritmo teria aplicabilidade a espaços de dimensão igual ou superior a dois. Desde o seu aparecimento, o Algoritmo de Graham tem contado com contribuições de vários autores, nomeadamente, com os trabalhos de Akl e Toussaint [2], publicado em 1978.

A estrutura geral deste algoritmo é de fácil compreensão e tem por base a Definição 3.3.11, isto é, parte do pressuposto de que para que l seja uma aresta extrema de um conjunto de pontos no plano é necessário e suficiente que todos os pontos do conjunto estejam num dos lados definidos por l.

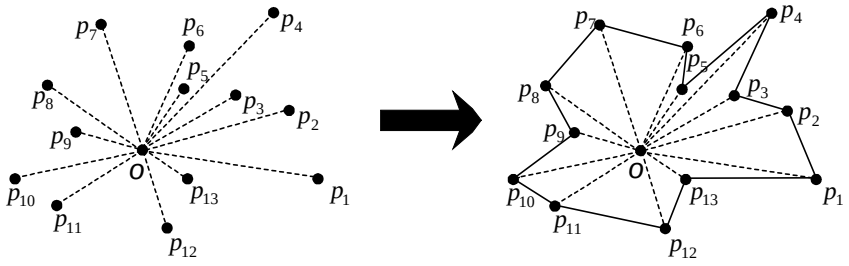


Figura 3.11: Obtenção de um polígono estrelado no Algoritmo de Graham.

O Algoritmo de Graham inicia-se com a escolha de um ponto interno O, que se considera a origem. Em seguida, usando o ponto O como *pivot*, ordenam-se lexicograficamente os pontos de $S \setminus \{O\}$ por ângulo polar (o ângulo formado pelo eixo horizontal e o raio

$[O, p))$ e distância à origem. Esta ordenação pode ser feita recorrendo a um qualquer algoritmo de ordenação, como por exemplo, o Algoritmo *Mergesort*.

Após a ordenação dos pontos do conjunto S obtém-se um polígono estrelado, como pode ser observado na Figura 3.11, ficando o nosso problema resumido ao de encontrar o invólucro convexo de um polígono estrelado. Neste caso, a ordem dos vértices do $CH(S)$ é dada pela ordenação polar. Assim, o problema resume-se ao de encontrar os pontos de S que são vértices do $CH(S)$.

O algoritmo segue com a análise de ternos de pontos consecutivos averiguando se os pontos de cada terno obedecem à primitiva *Left*, ou seja, verificando se estes têm ou não orientação positiva. Assim, se p_i, p_j e p_k forem três pontos que formam o triângulo $\Delta[p_i, p_j, p_k]$, se o $\Delta[p_i, p_j, p_k]$ tiver orientação positiva, p_j será candidato a ponto extremo. Se, pelo contrário, o triângulo tiver orientação negativa, o ponto p_j será eliminado, deixando de ser um candidato a ponto extremo.

O Algoritmo de Graham tem continuidade com a análise dos restantes ternos de pontos, em sentido anti-horário, determinando em cada iteração, se definem curva à esquerda ou curva à direita, até que todos os ternos de pontos consecutivos formem curva à esquerda. Por último, para obtermos o polígono pretendido basta unir todos os pontos não eliminados no procedimento anterior.

A implementação deste algoritmo obedece a alguns critérios no que diz respeito, por exemplo, à escolha do ponto que será tomado como origem. Assim, o ponto pode ser escolhido de forma a se encontrar no interior do invólucro, tal como se viu anteriormente, ou corresponder ao ponto de ordenada mínima.

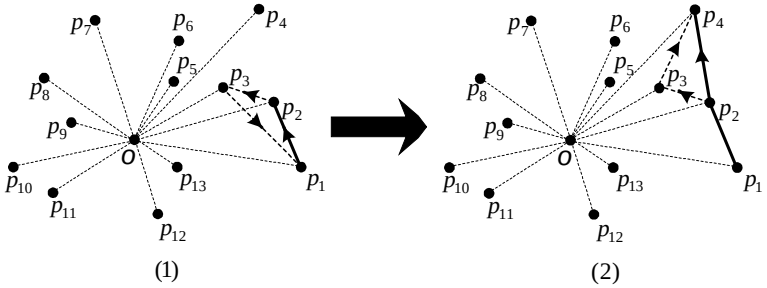


Figura 3.12: Algoritmo de Graham em acção.

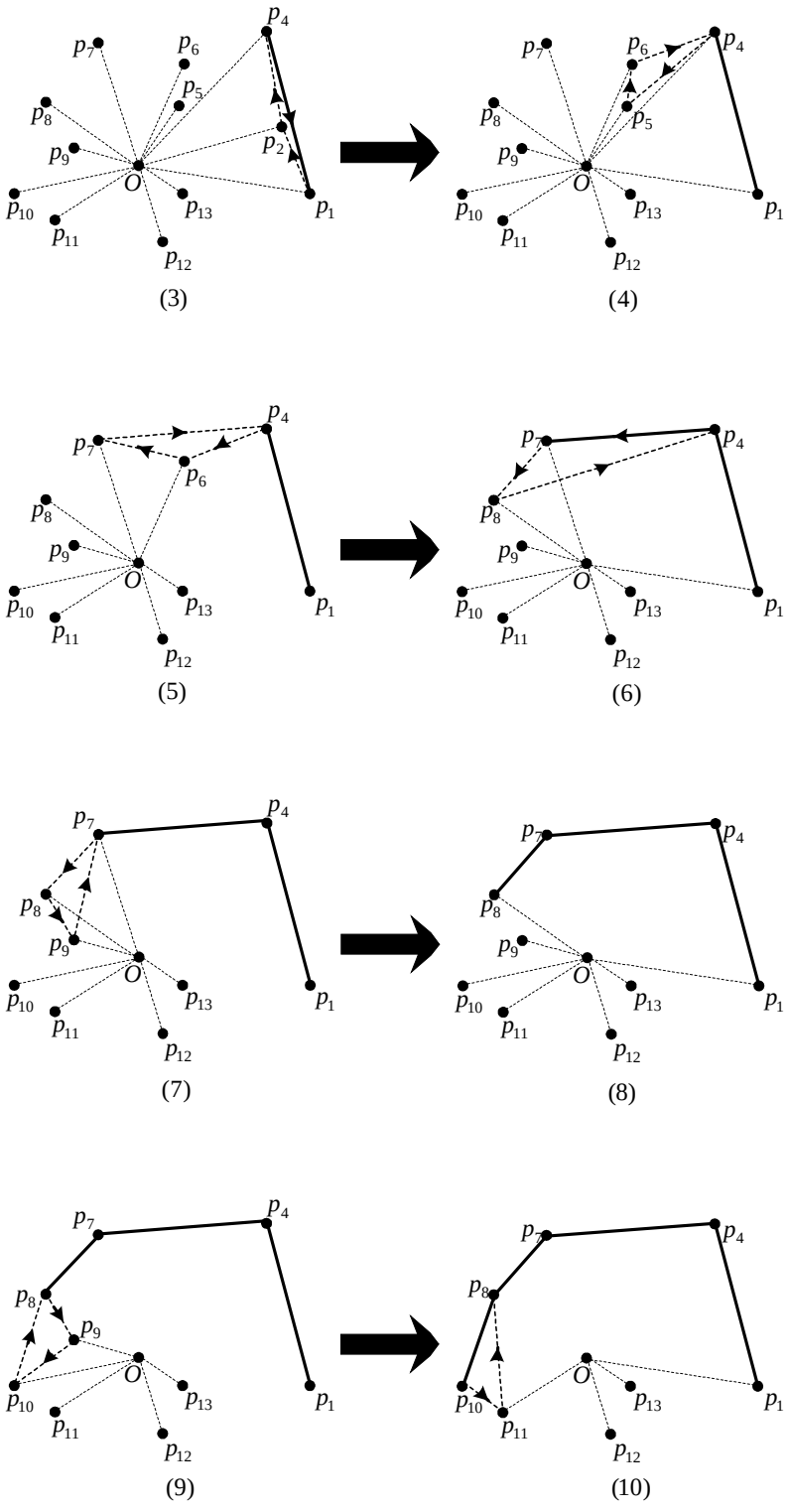


Figura 3.13: Algoritmo de Graham em acção.

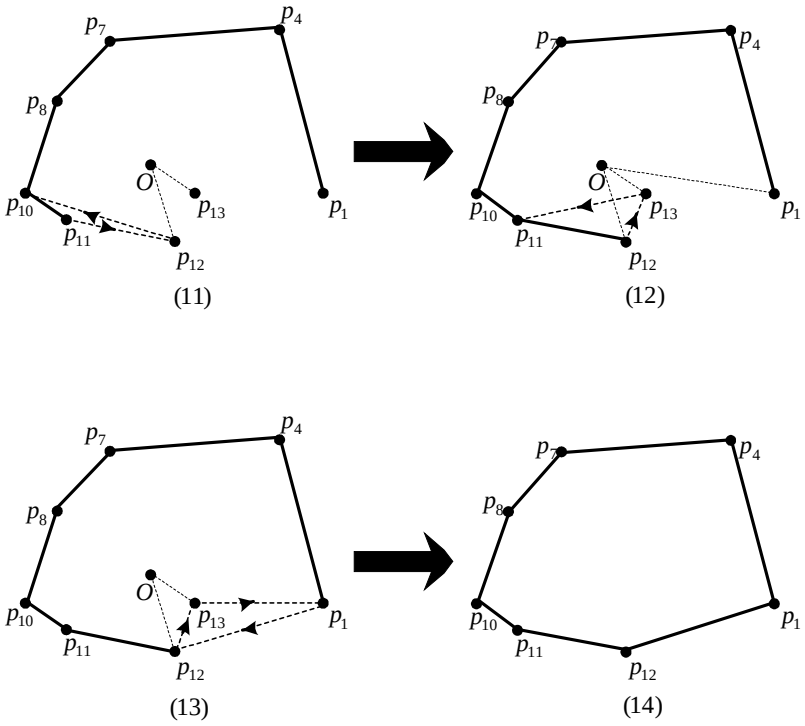


Figura 3.14: Algoritmo de Graham em acção.

Neste algoritmo, uma situação que pode trazer alguns problemas é o facto de poderem existir três ou mais pontos colineares, o que não foi assumido neste trabalho para facilitar a exposição de cada algoritmo. É certo, que tal facto afectaria certos aspectos deste algoritmo. Por exemplo, suponha-se que a um conjunto de pontos se aplica o algoritmo pretendendo como saída os pontos extremos do invólucro convexo. Sejam p_i, p_j e p_k três pontos consecutivos. Para verificarmos se tais pontos pertencem ao invólucro, basta ver se efectuam ou não curva à esquerda. O que pode ocorrer nesta situação é que p_i e p_j sejam colineares e, nesse caso, p_j será eliminado.

Uma outra situação delicada ocorre se, numa primeira fase do algoritmo, dois pontos a e b fazem o mesmo ângulo com o ponto *pivot* O. Perante esta situação podemos usar o seguinte critério: se $\angle a = \angle b$, então define-se que $a < b$ se $|a - O| < |b - O|$ e será eliminado o ponto a. Caso se inicie com os pontos p_i e p_j e p_j não é ponto extremo, o problema poderá ser resolvido com a escolha de $p_i - 1$ em vez de p_i. Esta solução tem por base o critério: se $\angle a = \angle b$ e $a < b$, de acordo com o critério anterior, então a

não é ponto extremo e, consequentemente, pode ser eliminado. Existe ainda um caso especial de colinearidade quando temos pontos coincidentes e, nesse caso, assumem-se os pontos como cópias de um ponto inicial permitindo a eliminação de tais pontos à excepção de um.

Observação 3.3.15 *No pseudo-código que se segue T designa uma pilha. Deste modo, T será uma estrutura de dados constituída por um conjunto ordenado de itens, no qual novos itens podem ser inseridos e a partir do qual podem ser eliminados itens de uma extremidade, chamada topo da pilha.*

Pseudo-código do Algoritmo de Graham (CH4)

Entrada: Um conjunto finito de pontos no plano, $S = \{p_1, ..., p_n\}$.

Saída: O invólucro convexo de S, $CH(S)$.

1. $T \leftarrow \emptyset$
2. Empilhar p_1, p_2 e p_3 na pilha T
3. $i \leftarrow 3$
4. Enquanto $i <= n$ fazer
5. $\quad p_t \leftarrow$ ponto do topo de T
6. $\quad p_{t-1} \leftarrow$ ponto abaixo de p_t em T
7. $\quad$ Se p_i está à esquerda de (p_{t-1}, p_t)
8. $\quad$ Então
9. $\quad\quad$ empilhar p_i em T
9. $\quad\quad$ $i \leftarrow i + 1$
10. $\quad$ Senão
11. $\quad\quad$ desempilhar p_t de T
12. Retornar T

Teorema 3.3.16 *O Algoritmo de Graham tem complexidade temporal $O(n \log n)$.*

Demonstração:

No Algoritmo de Graham a ordenação lexicográfica dos pontos é feita em $O(n\ log\ n)$ passos. Relativamente à análise efectuada a cada terno de pontos consecutivos, esta é

feita em $O(1)$. Os restantes passos deste algoritmo correm em tempo linear. Assim, este algoritmo irá dispender na sua totalidade de $O(n \log n)$. De salientar, que a maior parte do tempo complexidade gasto na execução deste algoritmo corresponde à fase de ordenação dos pontos. Feita esta ordenação, o Algoritmo de Graham corre em tempo linear. $\diamondsuit$

Apesar do Algoritmo de Graham ter uma maior eficiência do que os algoritmos apresentados anteriormente, continua a ser vantajosa a busca de algoritmos que permitam superar algumas das desvantagens deste algoritmo, tais como:

i) a aplicação do Algoritmo de Graham não poder ser generalizada a espaços de dimensão superior a dois;

ii) todos os pontos deverem estar disponíveis no início da execução do algoritmo;

iii) o uso de coordenadas polares pode envolver mudanças de variável embaraçosas em sistemas que possuem um número restrito de primitivas;

iv) o algoritmo não permite uma divisão do problema inicial em sub-problemas mais simples;

v) o algoritmo efectua a ordenação de todos os pontos independentemente do tamanho do invólucro;

vi) a ordenação dos pontos por ângulo polar envolve operações trigonométricas que, dependendo da sua complexidade, pode levar a um elevado desperdício de tempo real. Neste trabalho, considera-se que estas operações correm em tempo linear;

vii) se os pontos forem vértices de um polígono convexo, nenhum dos pontos será eliminado por este algoritmo;

viii) quando três pontos se encontram muito próximos, uma curva à esquerda pode ser facilmente confundida com uma curva à direita levando, por vezes, a erros quanto à eliminação de pontos extremos. Tal facto, pode ser superado considerando estes pontos distintos como sendo o mesmo ponto, através de um arredondamento.

3.3.5 Algoritmo da Cadeia Poligonal Monótona

O algoritmo proposto por Andrew's [3], em 1979, surgiu em alternativa ao Algoritmo de Graham por forma a minorar, em certas situações, a sua complexidade temporal. A vantagem deste algoritmo corresponde à fase da ordenação dos pontos onde, neste caso, é utilizada uma ordenação lexicográfica linear, que é mais eficiente do que a ordenação por ângulo polar usada no Algoritmo de Graham.
O Algoritmo da Cadeia Poligonal Monótona de Andrew's tem por base o *método da cadeia poligonal monótona*, apresentado por Lee e Preparata [33], em 1978.

O algoritmo começa por ordenar os pontos de $S = \{p_1, p_2, ..., p_n\}$ por incremento do valor de abcissa e, em seguida, por valor de ordenada. Sejam x_{min} e x_{max} os valores mínimo e o máximo de abcissa, dos pontos de S. Definam-se P_{--} como o ponto de abcissa x_{min} e com menor valor de ordenada, P_{-+} como o ponto de abcissa x_{min} e com maior valor de ordenada, P_{+-} como o ponto de abcissa x_{max} e o menor valor de ordenada e P_{++} como o ponto de abcissa x_{max} e o maior valor de ordenada. Note-se que quando existe um único ponto com valor de abcissa x_{min}, tem-se $P_{--} = P_{-+}$, assim como, quando existe um único ponto com x_{max} o ponto $P_{+-} = P_{++}$ (ver Figura 3.15). Em seguida, ligando os pontos P_{--} e P_{+-} é definida uma recta, seja l_{min}. De igual modo, usando os pontos P_{++} e P_{-+} define-se uma outra recta l_{max}.

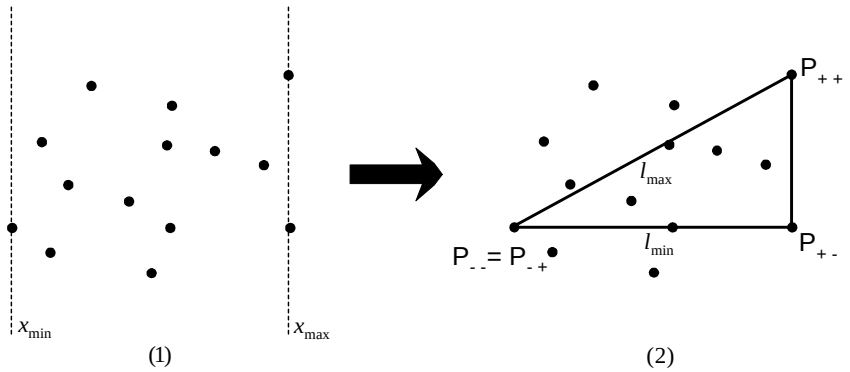


Figura 3.15: Algoritmo da Cadeia Poligonal Monótona em acção.

O algoritmo segue com a construção de uma cadeia poligonal convexa definida na parte inferior da recta l_{min} e que une os pontos p_{--} e p_{+-}, seja Ω_{min} esta cadeia. Do mesmo modo, é construída uma outra cadeia poligonal convexa situada na parte superior da recta l_{max} e que une os pontos p_{-+} e p_{++}, seja Ω_{max} esta nova cadeia. Estas duas cadeias poligonais monótonas são construídas recorrendo ao mesmo método que é usado no Algoritmo de Graham, quando se pretende analisar os ternos de pontos.
Finalmente, o invólucro convexo será construído unindo as duas cadeias poligonais convexas Ω_{min} e Ω_{max}.

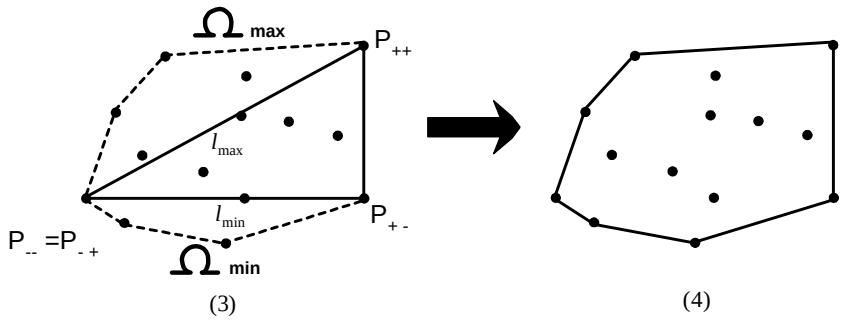


Figura 3.16: Algoritmo da Cadeia Poligonal Monótona em acção.

Pseudo-código do Algoritmo da Cadeia Poligonal Monótona (CH5)

Entrada: Um conjunto finito de pontos no plano, $S = \{p_1, ..., p_n\}$.

Saída: O invólucro convexo de S, $CH(S)$.

1. Remover os pontos que têm coordenadas iguais, obtendo o conjunto $\overline{S}$.

2. Ordenar os pontos de $\overline{S}$ por valor de abcissa (em caso de empate, ordenar por valor de ordenada).

3. Inicializar uma lista U, que contém os vértices da cadeia poligonal superior.

4. Inicializar uma lista L, que contém os vértices da cadeia poligonal inferior.

5. Para $i = 1$ até n fazer

    Enquanto L contiver pelo menos dois pontos e a sequência dos últimos dois pontos em L e $\overline{S}[i]$ não fizer curva à esquerda

Remover o último ponto de L

Acrescentar $\overline{S}[i]$ a L.

6. Para $i = n$ até 1 fazer

Enquanto U contiver pelo menos dois pontos e a sequência dos últimos

dois pontos em U e $\overline{S}[i]$ não fizer curva à esquerda

Remover o último ponto de U

Acrescentar $\overline{S}[i]$ a U.

7. Remover o primeiro ponto da lista U e o último da lista L.

8. Concatenar L e U para obter $CH(S) = L \cup U$.

Teorema 3.3.17 *O Algoritmo da Cadeia Poligonal Monótona tem complexidade temporal* $O(n \log n)$.

Demonstração: Tal como sucede com o Algoritmo de Graham, este algoritmo necessita, numa primeira fase, correspondente à ordenação lexicográfica dos pontos, de $O(n \log n)$ operações. A análise efectuada aos ternos de pontos que constituem cada uma das cadeias poligonais é feita em $O(n)$. Assim, para as duas cadeias poligonais, Ω_{min} e Ω_{max}, teremos complexidade $O(2n) = O(n)$. ♢

3.3.6 Divisão e Conquista

Os algoritmos apresentados, em seguida, baseiam-se no paradigma dividir para conquistar. Este paradigma é vulgarmente usado para resolver alguns problemas clássicos em Geometria Computacional e consiste na divisão de problemas complexos em problemas mais simples. O método da divisão e conquista é considerado, tal como já foi referido, um método recursivo, ou seja, começa por resolver cada um dos sub-problemas mais simples combinando, em seguida, as suas soluções com vista à resolução do problema inicial. Este método encontra-se generalizado no algoritmo seguinte.

Algoritmo Divisão e Conquista

Entrada: Um problema P de tamanho n.

Saída: Solução de P.

1. Se $n = 1$, então resolva o problema P e pare.
2. Se $n > 1$, então divida o problema P em k sub-problemas de tamanho $\frac{n}{k}$.
3. Resolva cada sub-problema recursivamente.
4. Combine as soluções dos vários sub-problemas para obter a solução do problema inicial P.

A complexidade de um algoritmo que recorra ao método da divisão e conquista é variável dependendo, como no caso dos invólucros convexos, não só do tamanho do conjunto de entrada, mas também da posição assumida por cada ponto do conjunto no plano.

Seja $T(n)$ a complexidade temporal de um algoritmo que resolve um problema de tamanho n. Podemos assumir, sem perda de generalidade, que se o conjunto tiver tamanho 1, então $T(1) = c$, ou seja, o algoritmo irá correr em tempo linear. No que diz respeito aos passos 2 e 3, para um conjunto de n pontos será necessário nc_1, onde c_1 é uma constante. Ao dividirmos o problema inicial de tamanho n em k sub-problemas, iremos ter o problema reduzido a sub-problemas de tamanho $(\frac{n}{k})$, cuja complexidade temporal será $T(\frac{n}{k})$. Assim, como no segundo passo os problemas serão resolvidos em $c_1T(\frac{n}{k})$, a complexidade temporal de qualquer algoritmo de divisão e conquista pode ser calculada recursivamente por:

$$\begin{cases} T(1) = c \\ T(n) = kT(\frac{n}{k}) + cn \end{cases}$$

Teorema 3.3.18 *Se a fase da divisão correr em tempo linear, então um algoritmo de divisão e conquista será executado em $T(n) = O(n \log n)$ tempo complexidade.*

3.3.6.1 Algoritmo Quickhull

O algoritmo seguinte foi proposto independentemente por vários autores : Eddy [17], Bykat [7], Green e Silverman [27]. Mais tarde, veio a ser baptizado por Preparata e Shamos [40] de **Quickhull**, visto se tratar de uma extensão do algoritmo de ordenação Quicksort. O Algoritmo Quickhull tem por base o paradigma da divisão e conquista sendo, portanto, um algoritmo recursivo que centra todas as suas atenções nos pontos da fronteira do conjunto.

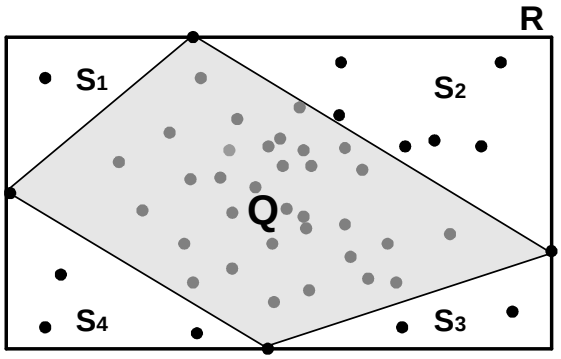


Figura 3.17: Pré-processamento dos quatro pontos extremos.

O Algoritmo Quickhull pode iniciar-se de diversas formas. Assim sendo, escolhemos um procedimento simples e interessante que tem por base o Algoritmo proposto por Akl e Toussaint. Este algoritmo será apresentado de uma forma mais cuidada na subsecção 3.3.7.

Seja S um conjunto de n pontos no plano. O Algoritmo Quickhull inicia-se com um pré-processamento para a escolha de quatro pontos extremos distintos. Destes quatro pontos, dois irão corresponder aos valores extremos de abcissa (valor máximo e valor mínimo) e os outros dois pontos corresponderão aos valores extremos de ordenada (valor máximo e valor mínimo). Ao efectuarmos a selecção destes quatro pontos pode ocorrer um empate, por exemplo, dois pontos com a mesma abcissa e, nesse caso, podemos

optar pela escolha dos pontos com menor valor de abcissa. Caso se verifique um empate quanto ao valor de ordenada pode optar-se pelo ponto com menor valor de ordenada, por exemplo.

Unindo os quatro pontos seleccionados obtemos um rectângulo $R := [X_{min}, X_{max}] \times$ $\times [Y_{min}, Y_{max}]$ onde S está descrito, e um quadrilátero Q cujos vértices são pontos extremos que irão pertencer ao invólucro convexo de S (ver Figura 3.17). Os pontos que se encontram no interior deste quadrilátero Q podem ser eliminados, assim como, os pontos que se encontram sobre os quatro segmentos de recta que o definem, à excepção dos próprios vértices do polígono. Uma vez que os vértices do quadrilátero Q pertencem aos lados de R, o conjunto $R \setminus Q$ de $\mathbb{R}^2$ pode ser considerado como a reunião de quatro regiões distintas: S_1, S_2, S_3 e S_4.

O procedimento anterior conduz-nos de um problema inicial complexo para quatro sub-problemas mais simples, cada um dos quais referente a uma região. Estes problemas serão agora resolvidos independentemente sendo encontrada, para cada um deles, uma cadeia poligonal. Por fim, concatenando as quatro cadeias poligonais obteremos o invólucro convexo pretendido. Vejamos, agora, como se processa a resolução do problema correspondente a cada região:

Seja S_1 o conjunto dos pontos correspondentes à primeira região e a e b os pontos extremos que irão constituir a base de um triângulo cujo terceiro vértice será um ponto extremo, algoritmicamente seleccionado, que se encontra na direcção da recta ortogonal ao segmento $[a, b]$. A fase seguinte corresponde à eliminacão dos pontos que se encontram no interior do triângulo $\Delta[a, b, c]$. O algoritmo é chamado recursivamente até que mais nenhum ponto seja eliminado.

O procedimento anteriormente descrito é aplicado às restantes três regiões S_2, S_3 e S_4.

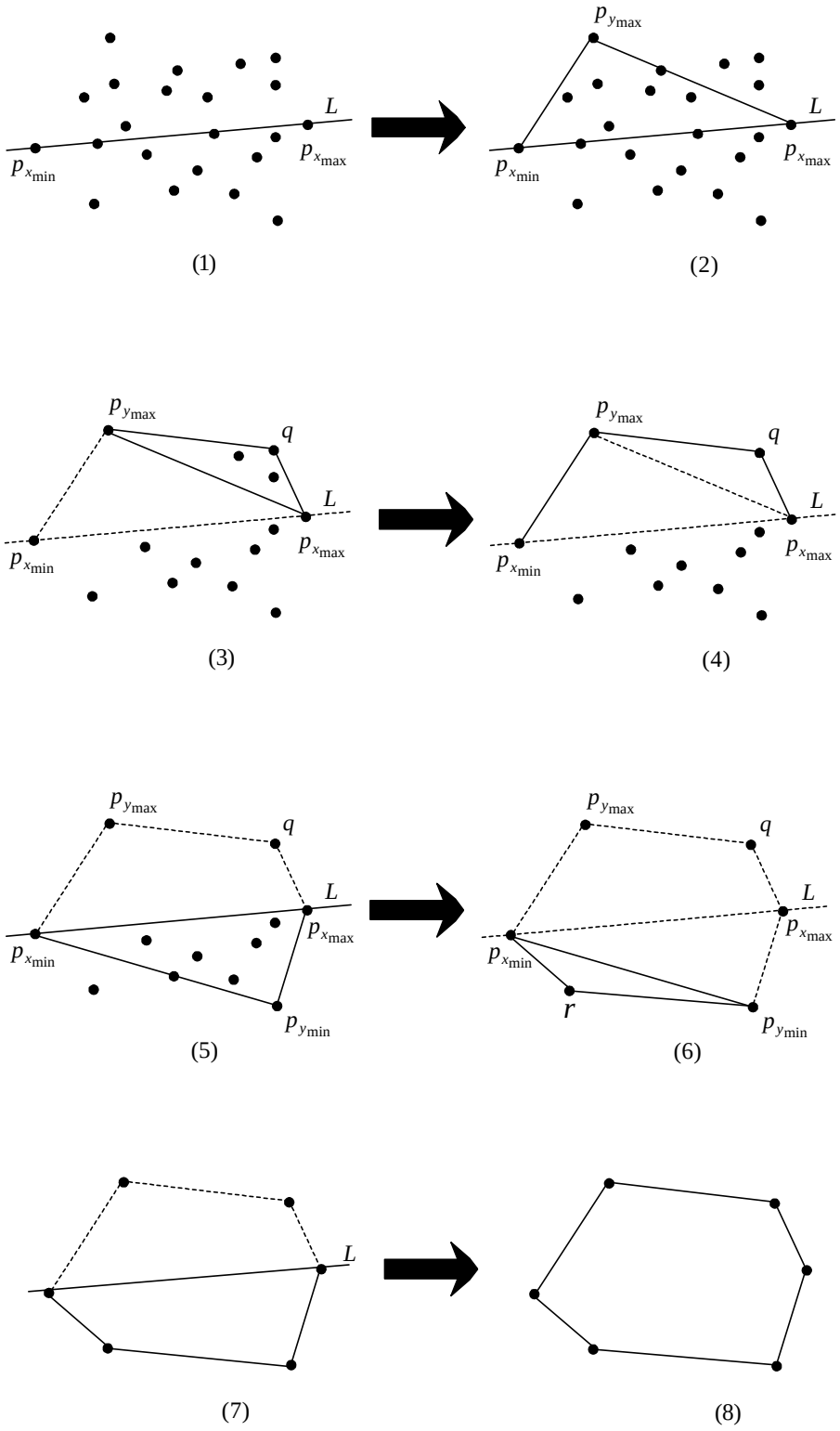


Figura 3.18: Algoritmo Quickhull em acção.

Pseudo-código do Algoritmo QuickHull (CH6)

Entrada: Dois pontos distintos do plano a e b e um conjunto de pontos S de tal forma que qualquer ponto de S se encontra num dos lados definidos pela recta que passa pelos pontos a e b.

Saída: O invólucro convexo de S, $CH(S)$.

1. Se $S = \emptyset$ então retornar $[a, b]$

2. Senão

3. $\quad c \leftarrow$ ponto de distância máxima ao segmento $[a, b]$

4. $\quad S_1 \leftarrow$ pontos que se encontram do lado oposto ao lado onde está o ponto b relativamente à recta L_2 passando pelo segmento $[a, c]$

5. $\quad S_2 \leftarrow$ pontos que se encontram do lado oposto ao lado onde está o ponto a relativamente à recta L_1 passando pelo segmento $[b, c]$

6. $\quad$ Retornar Quickhull (a, c, S_1) concatenando com Quickhull (c, b, S_2)

Teorema 3.3.19 *O Algoritmo Quickhull tem complexidade temporal* $O(n \log n)$.

Demonstração: A complexidade deste algoritmo depende, tal como acontece noutros algoritmos, da disposição dos pontos do conjunto de entrada. Assim, se os pontos se encontrarem dispostos em circunferência tornar-se-á um algoritmo de complexidade temporal $O(n^2)$ e, por isso, pouco eficiente. Este facto, deve-se em parte à não ocorrência de eliminação de pontos na fase inicial deste algoritmo. A complexidade do algoritmo Quickhull depende também do número de pontos do conjunto S_1 e S_2. Sejam $\alpha_1 + \alpha_2 = n$, onde $|S_1| = \alpha_1$ e $|S_2| = \alpha_2$. Se denotarmos por $T(n)$ a função complexidade de tempo do Algoritmo Quickhull, se tivermos $\alpha_1 = 1$ e $\alpha_2 = n - 1$ teremos que $T(n) \leq T(n-1) + cn = T(n-2) + c(n-1) + cn$ pelo que, repetindo a expansão, se tem $T(n) \leq c + c2 + ... + c(n-2) + c(n-1) + cn = O(n^2)$.
No entanto, caso geral, este algoritmo é eficiente e para encontrar o invólucro convexo de um conjunto S com n pontos teremos $T(n) \leq T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + cn = O(n \log n)$.

$\diamondsuit$

3.3.6.2 Algoritmo Mergehull

Seja S um conjunto de n pontos no plano. Tal como sucede em qualquer algoritmo que utilize o método da divisão e conquista, o Algoritmo Mergehull desenvolve-se fundamentalmente em três fases distintas. Ele inicia-se com a divisão de S em dois subconjuntos S_1 e S_2, com o mesmo número de elementos. Procedendo, em seguida, à determinação independente dos invólucros convexos de S_1 e S_2. Finalmente, obtém-se o $CH(S)$ concatenando $CH(S_1)$ e $CH(S_2)$.

No que se refere à fase da "divisão"de S em S_1 e S_2, esta pode ser executada recorrendo a dois processos diferentes. O primeiro consiste em separar S_1 de S_2 através de um ponto P que possui o valor de abcissa médio. No segundo, efectua-se uma pré-ordenação dos pontos de S segundo os valores de abcissa, armazenando-os numa lista e a partir daí extrai-se o valor médio da lista, o que é feito em tempo linear. Nesta etapa, pode ocorrer que dois ou mais pontos tenham o mesmo valor de abcissa. Nesse caso, considera-se a ordenação desses pontos pelo seu valor de ordenada. A fase seguinte corresponde à fase da "conquista"e é nela que se constroem os invólucros convexos dos dois conjuntos determinados na etapa anterior, ou seja, $CH(S_1)$ e $CH(S_2)$.

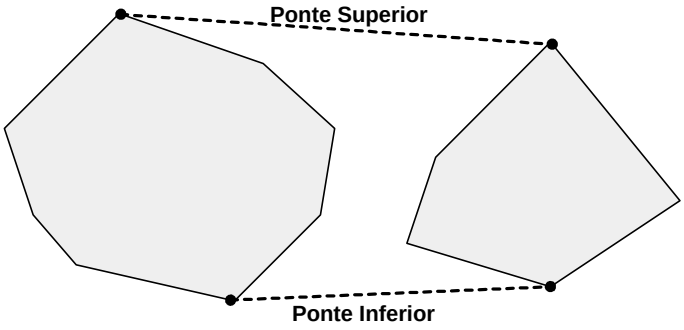


Figura 3.19: Ponte Superior e Ponte Inferior.

O $CH(S)$ só fica construído depois de concatenar $CH(S_1)$ com o $CH(S_2)$. Para tal é necessário encontrar a chamada *ponte superior* que corresponde à recta tangente aos pontos com maior valor de ordenada de $CH(S_1)$ e $CH(S_2)$ e a *ponte inferior* que

corresponde à recta tangente aos pontos de $CH(S_1)$ e $CH(S_2)$ cuja ordenada é mínima (ver Figura 3.19).

Para a construção das pontes superior e inferior recorre-se a dois algoritmos auxiliares ao Algoritmo Mergehull: o *Algoritmo Ponte Superior* e o *Algoritmo Ponte Inferior*.

Algoritmos Auxiliares ao Algoritmo Mergehull

Algoritmo Ponte Superior

Entrada: Dois invólucros convexos $CH(S_1)$ e $CH(S_2)$ que se encontram separados por uma recta vertical de tal forma que $CH(S_1)$ se encontra à esquerda da recta e $CH(S_2)$ à direita.
Saída: A ponte superior de $CH(S_1)$ e $CH(S_2)$.

1. Seja p o ponto de $CH(S_1)$ com menor valor de abcissa e q o ponto de $CH(S_2)$ com maior valor de abcissa.
2. Seja L a recta que une os pontos p e q.
3. Enquanto L não é ponte superior fazer
4. Enquanto existir um ponto $p^{'}$ em $CH(S_1)$ acima da recta L, substituir p por $p^{'}$ e traçar uma nova recta L
5. Enquanto existir em $CH(S_2)$ um ponto $q^{'}$ que esteja acima da recta L, substituir q por $q^{'}$ e traçar uma nova recta L

Observação 3.3.20 *O algoritmo usado na construção da ponte inferior designa-se por* ***Algoritmo Ponte Inferior*** *e é análogo ao algoritmo anterior e, por isso, não será aqui exposto.*

Depois de estarem construídas as pontes superior e inferior, considerem-se em S_1 os pontos $S(S_1)$ e $I(S_1)$ que pertencem, respectivamente, às pontes superior e inferior.

Todos os vértices de $CH(S_1)$ que se encontrarem dispostos em sentido horário de $S(S_1)$ para $I(S_1)$ serão eliminados. Este raciocínio será também aplicado a $CH(S_2)$. Obtendo-se, assim, apenas os vértices pertencentes ao $CH(S)$, tal como era pretendido.

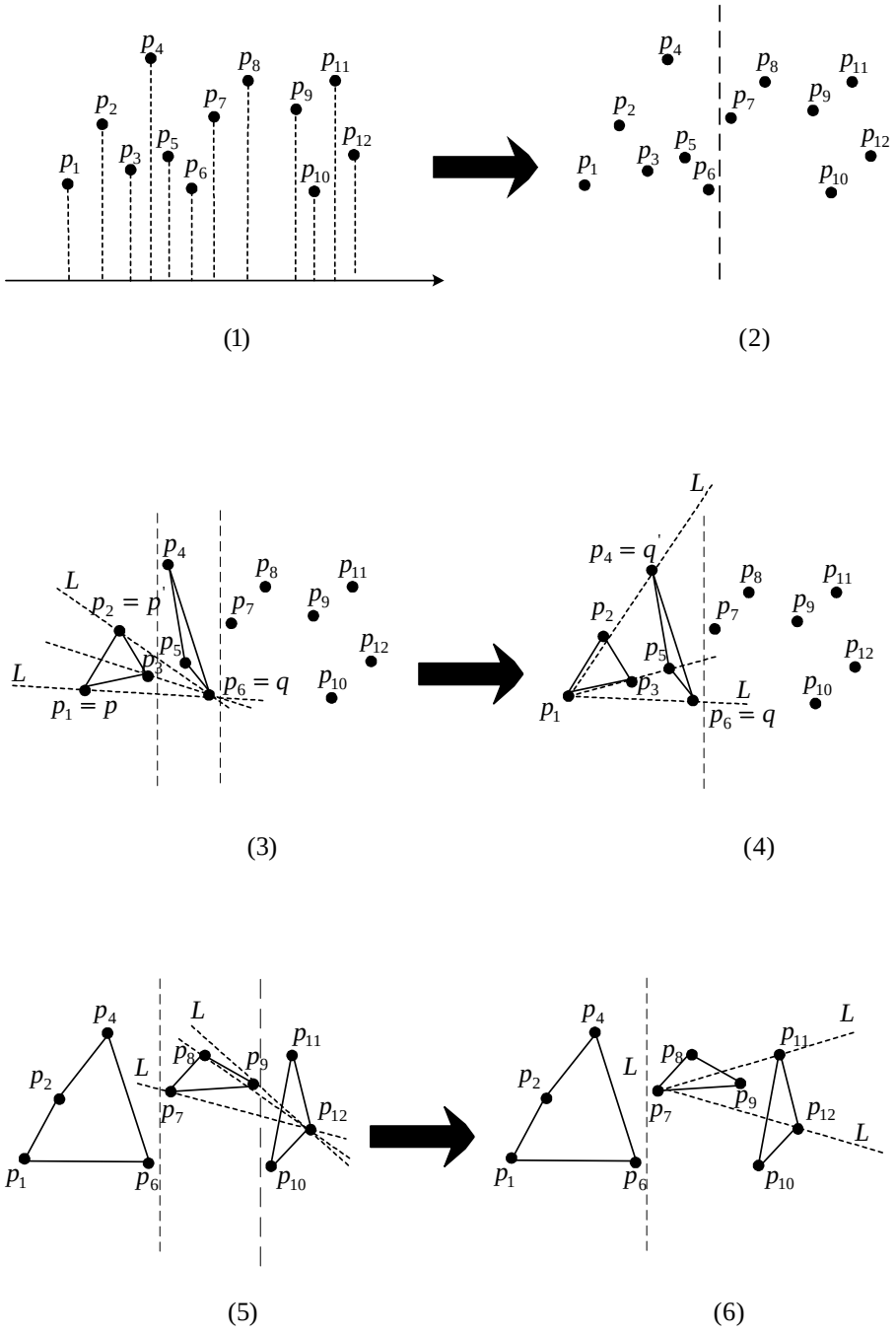
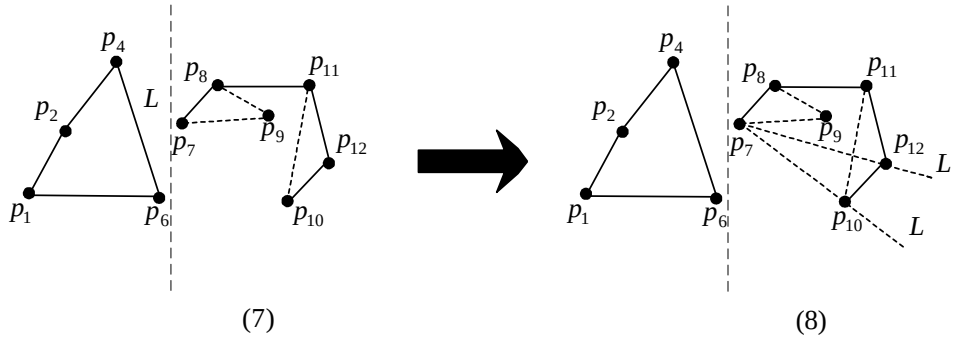
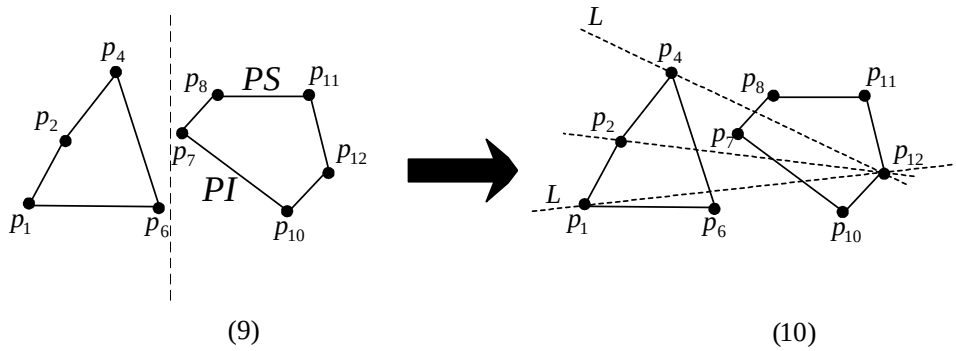


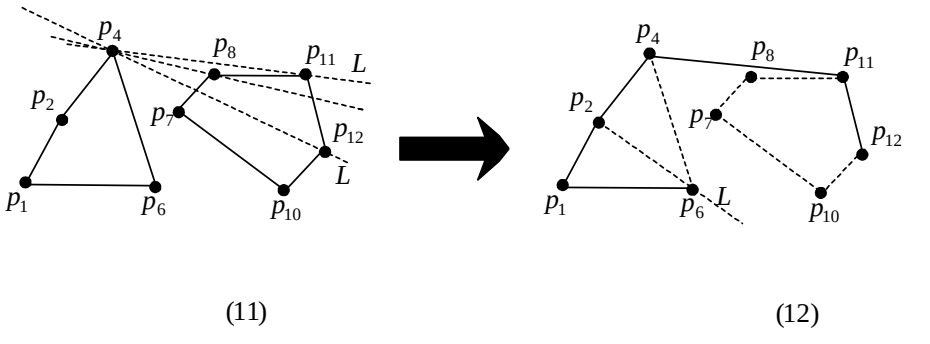
Figura 3.20: Algoritmo Mergehull em acção.



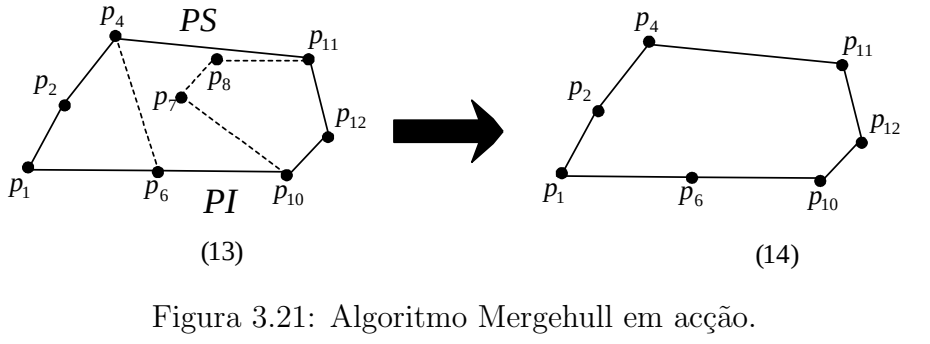
(7) (8)



(9) (10)



(11) (12)



(13) (14)

Figura 3.21: Algoritmo Mergehull em acção.

Pseudo-código do Algoritmo Mergehull (CH7)

Entrada: Um conjunto finito de pontos no plano, $S = \{p_1, ..., p_n\}$.

Saída: O invólucro convexo de S, $CH(S)$.

1. Ordenar os pontos de S por valores das abcissas.

2. Se n é ímpar: $S_1 = \{p_1, ..., p_{\frac{n-1}{2}}\}$ e $S_2 = \{p_{\frac{n-1}{2}+1}, ..., p_n\}$.

3. Se n é par: $S_1 = \{p_1, ..., p_{\frac{n}{2}}\}$ e $S_2 = \{p_{\frac{n}{2}+1}, ..., p_n\}$.

4. Construir recursivamente os invólucros convexos $CH(S_1)$ e $CH(S_2)$.

5. Seja p_{max} o vértice de maior valor de abcissa de $CH(S_1)$ e Q_{min} o vértice com menor valor de abcissa de $CH(S_2)$.

6. Seja L a recta que passa pelos pontos p_{max} e Q_{min}.

7. Construir a ponte superior.

8. Construir a ponte inferior.

9. Eliminar os pontos de vértices $(S_1) \cup$ vértices (S_2) que surgem em sentido anti-horário quando se passa de $S(S_1)$ para $I(S_1)$. Do mesmo modo, eliminar os pontos de vértices $(S_1) \cup$ vértices (S_2) que surgem em sentido anti-horário quando se passa de $S(S_2)$ para $I(S_2)$.

Teorema 3.3.21 *O Algoritmo Mergehull tem complexidade temporal* $O(n \log n)$.

Demonstração: Para se encontrar o invólucro convexo de S ter-se-á, em termos de complexidade, tal como sucede em qualquer algoritmo que recorra à divisão e conquista, $T(n) \leq T(\alpha_1) + T(\alpha_2) + cn$, onde c é uma constante e cn a parcela associada ao número de operações efectuadas pelo algoritmo para separar S em S_1 e S_2. Ora, S_1 e S_2 terão de ter, se possível, a mesma dimensão. Note-se que $\alpha_1 + \alpha_2 \leq n$. Assim, quando $\alpha_1 + \alpha_2 = n$ teremos que $|S_1| = \alpha_1$ e $|S_2| = \alpha_2$. Logo, tal como acontece no Algoritmo Quickhull, tem-se: $T(n) \leq T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + cn = O(n\, log\, n)$. $\diamondsuit$

Nos dois algoritmos anteriores, verificamos que a etapa correspondente à combinação tem uma importância relativa. No caso do Algoritmo Quickhull é uma etapa trivial. Contudo, o mesmo não acontece no Algoritmo Mergehull, uma vez que é nesta etapa que são concatenados os invólucros convexos $CH(S_1)$ e $CH(S_2)$, para a construção de $CH(S)$, o que obriga a um maior esforço.
Este algoritmo, fundamenta-se na seguinte relação:

$$CH(S_1 \cup S_2) = CH(CH(S_1) \cup CH(S_2))$$

Note-se que aos conjuntos $CH(S_1)$ e $CH(S_2)$ correspondem a polígonos convexos. Este facto conduz-nos a um novo problema.

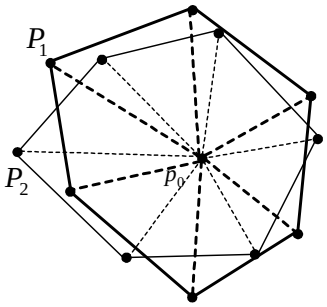
Invólucro Convexo da União de Polígonos Convexos: sejam P_1 e P_2 dois polígonos convexos. Encontrar o invólucro convexo da sua união.

Para que se possa garantir a eficiência de qualquer algoritmo que recorra ao paradigma da divisão e conquista é fundamental que a solução de cada sub-problema seja rapidamente produzida. Caso tal não aconteça, a complexidade do algoritmo principal poderá não corresponder à pretendida.

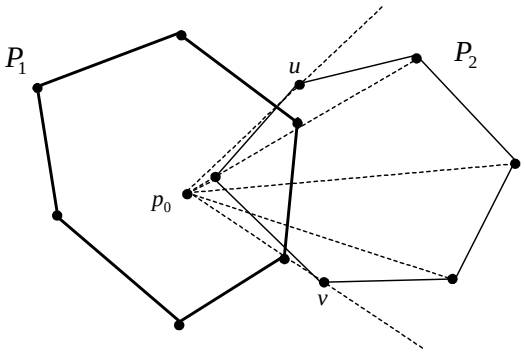
Note-se que os conjuntos $CH(S_1)$ e $CH(S_2)$ podem ser obtidos com complexidade temporal de $O(n \log n)$ recorrendo a um qualquer algoritmo para o efeito, como por exemplo, o Algoritmo de Graham. Relembre-se que neste algoritmo a maior parcela de tempo dispendido corresponde à fase em que os pontos são ordenados por ângulo polar em torno de um ponto interior p_0. Usemos, então, o Algoritmo de Graham para construir $CH(S_1)$ e $CH(S_2)$. Neste caso, ele terá uma complexidade temporal óptima uma vez que aos conjuntos $CH(S_1)$ e $CH(S_2)$ correspondem dois polígonos convexos, P_1 e P_2, que já se encontram ordenados relativamente a qualquer ponto interior.
Para solucionarmos o problema do invólucro convexo da união de polígonos convexos, considerem-se os polígonos P_1 e P_2, em que os seus vértices já se encontram ordenados em sentido positivo e um ponto p_0 interior a um dos polígonos, por exemplo, P_1. Considere-se p_0 como sendo, por exemplo, o centroide de três quaisquer vértices de P_1.

Perante esta situação é possível distinguir dois casos:

i) p_0 encontra-se no interior de P_1 e P_2



ii) p_0 encontra-se apenas no interior de P_1



No primeiro caso, P_1 e P_2 já se encontram ordenados por ângulo polar em torno de p_0. É fácil, com complexidade temporal linear, armazenar os pontos de P_1 e de P_2 numa lista ordenada comum e temos assim o problema solucionado. Relativamente ao segundo caso, P_2 encontra-se contido num ângulo de vértice p_0, cujos lados são as semi-rectas $[p_0, u)$ e $[p_0, v)$. Note-se que os vértices u e v separam os vértices de P_2 em duas sub-listas, em que uma delas encontra-se ordenada angularmente em torno de p_0. Observe-se que os pontos de uma destas listas encontram-se no interior do triângulo $\Delta[p_0uv]$ e podem, por isso, ser eliminados. Assim, ficaremos apenas com uma sub-lista que terá de ser combinada com a lista de vértices de P_1, o que leva tempo $O(n)$.

O modo como são determinados os vértices u e v é simples, bastando para tal efectuar uma análise sob o ponto de vista da monotonia da sequência de valores do ângulo orientado nos vértices de P_2. Assim, encontraremos u e v sempre que houver uma mudança dessa mesma monotonia.

O algoritmo para determinar o invólucro convexo da união de dois polígonos convexos foi proposto por M. I. Shamos [43] e assenta nas seguintes instruções:

1. Seleccionar um ponto p_0 interior ao polígono P_1. Este ponto será também interior ao $CH(P_1 \cup P_2)$.

2. Na lista onde se encontram armazenados os vértices de P_2 procurar uma sub-lista de extremos u e v, onde todos os pontos se encontrem armazenados angularmente em torno de p_0. Note-se que se p_0 se encontrar no interior de P_2 encontraremos, nessa mesma lista, todos os vértices de P_2.

3. Combinar a lista de vértices de P_1 com a encontrada no passo anterior, armazenando--as numa lista única, onde todos os vértices se encontram angularmente ordenados em torno de p_0.

4. Aplicar o Algoritmo de Graham a esta lista. Este algoritmo não será aplicado na sua totalidade uma vez que os pontos já se encontram ordenados.

Quanto à complexidade deste algoritmo, podemos referir que a cada passo corresponde uma complexidade linear, o que fundamenta o resultado seguinte:

Teorema 3.3.22 *O invólucro convexo da união de dois polígonos convexos P_1 e P_2 pode ser determinado em tempo proporcional ao número total de vértices de P_1 e P_2.*

Se P_1 tiver m vértices e P_2 n vértices, este algoritmo terá uma complexidade temporal de $O(m+n)$, o que é óptimo!

3.3.7 Algoritmo proposto por Akl e Toussaint

O algoritmo seguinte constituí uma referência em termos de algoritmos que constroem invólucros convexos. Foi proposto por Selim G. Akl e Godfried Toussaint [2], em 1978. Este algoritmo inicia-se com a identificação de quatro pontos extremos correspondentes aos pontos com valores extremos de abcissa e ordenada, respectivamente, X_{max}, X_{min}, Y_{max} e Y_{min}. Estes são pontos que definem um quadrilátero e que irão pertencer, garantidamente, ao invólucro convexo do conjunto. O mesmo não acontece com os pontos que se encontram no seu interior ou sobre as arestas que o definem, que serão eliminados ficando o conjunto inicial dividido em quatro subconjuntos correspondentes a quatro regiões distintas, tal como se pode observar na Figura 3.22.

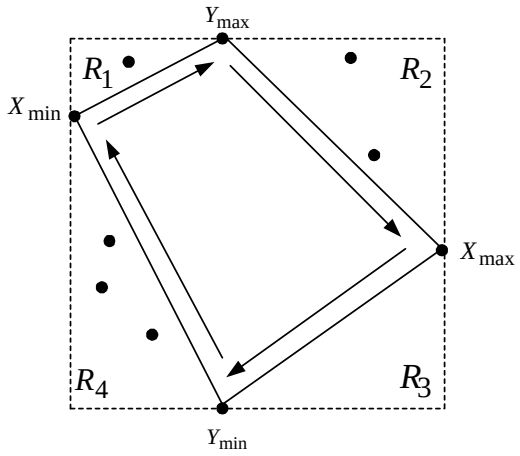


Figura 3.22: Fase inicial do algoritmo.

Por vezes, há pontos extremos que são coincidentes e esse facto faz com que o número de regiões possa ser variável e inferior a quatro. Assim, se dois dos quatro pontos extremos de ordenada e de abcissa forem coincidentes teremos um total de três regiões distintas. Podemos ainda identificar uma outra situação correspondente ao caso em que temos dois pares desses pontos coincidentes. Perante tal situação teremos apenas duas regiões, como pode ser observado na Figura 3.23.

O algoritmo segue com a ordenação dos pontos do conjunto pelo respectivo valor de abcissa. Da região 1 para a região 2 efectua-se uma ordenação no sentido ascendente e, da região 3 para a região 4, a ordenação é feita de forma descendente.

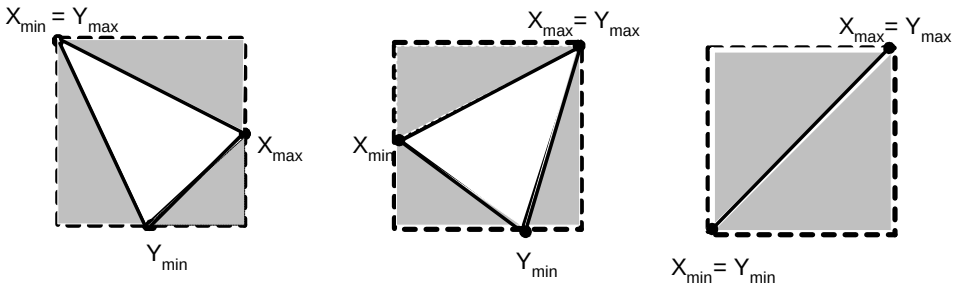


Figura 3.23: Divisão em menos do que quatro regiões.

Seguidamente, o objectivo é encontrar um caminho que una um dos quatro pontos extremos inicialmente determinados a outro ponto extremo da mesma região. Para tal, efectua-se uma análise aos ternos de pontos consecutivos de cada região, sejam p_k, p_{k+1} e p_{k+2}, verificando o sinal do produto externo entre o vector que liga os pontos p_k e p_{k+1} e o vector de extremidades p_{k+1} e p_{k+2}. Se o sinal do produto externo for negativo assume-se o ponto p_{k+1}. Caso contrário, o ponto é eliminado.

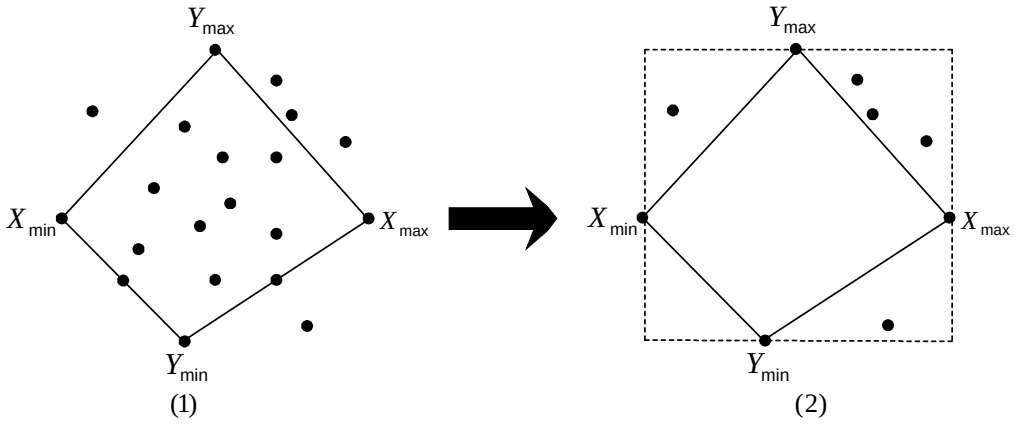


Figura 3.24: Algoritmo proposto por Akl e Toussaint em acção.

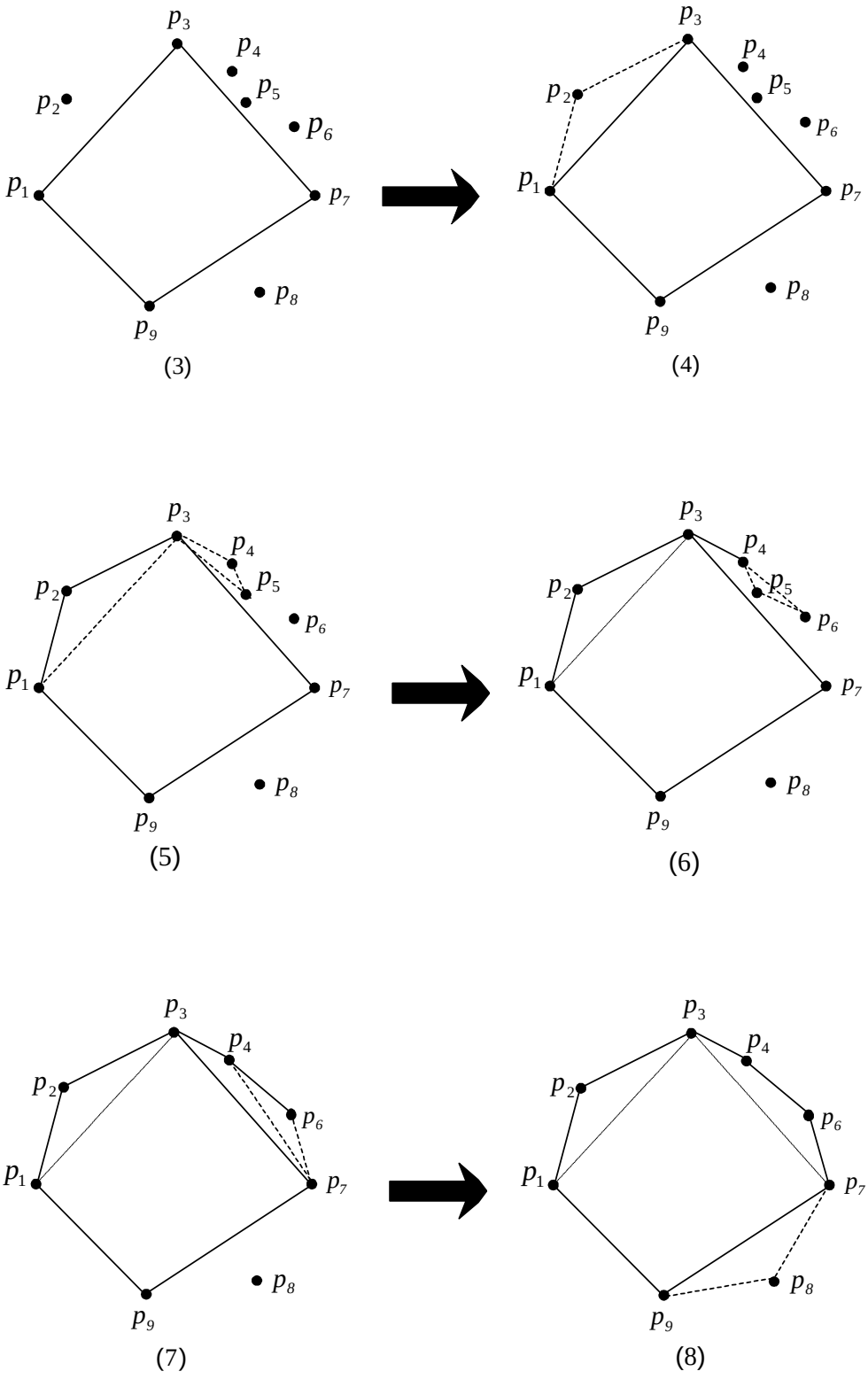


Figura 3.25: Algoritmo de Akl e Toussaint em acção.

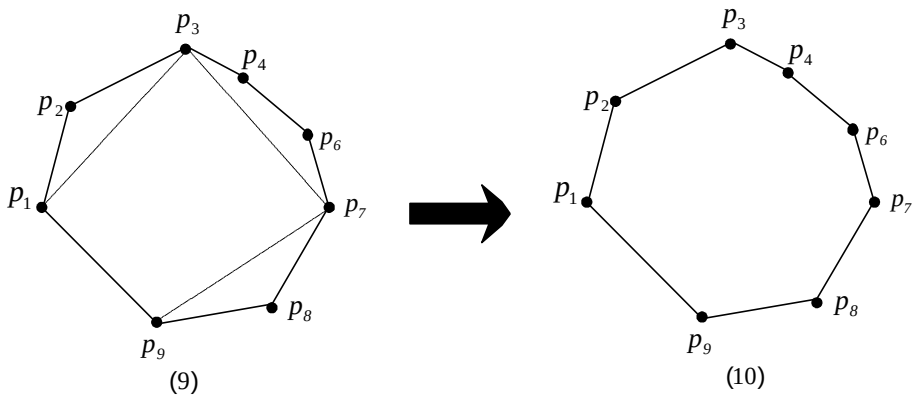


Figura 3.26: Algoritmo de Akl e Toussaint em acção.

Pseudo-código do Algoritmo proposto por AKL e Toussaint (CH8)

Entrada: Um conjunto finito de pontos no plano, $S = \{p_1, ..., p_n\}$.

Saída: O invólucro convexo de S, $CH(S)$.

1. Encontrar os quatro pontos extremos e eliminar os pontos que se encontram no interior do polígono por eles definido.

2. Ordenar os restantes pontos do conjunto por valor de abcissa: da esquerda para a direita (da região 1 para a região 2) e da direita para a esquerda (da região 3 para a região 4).

3. Para cada uma das regiões encontrar a cadeia convexa de um ponto extremo para outro usando as regras seguintes:

4. Começar com um ponto extremo e fazer (a) e (b) abaixo para quaisquer três pontos consecutivos p_k, p_{k+1}, p_{k+2}, até que o outro ponto extremo seja alcançado.

   (a) Computar $CH(S)$

   (b) Se $S \geq 0$ avançar um ponto. Senão, eliminar o ponto P_{k+1} e recuar um ponto

5. Se 4. ficar completo sem que nenhum ponto seja eliminado parar. Senão, repetir 4.

Teorema 3.3.23 *O Algoritmo proposto por Akl e Toussaint tem complexidade temporal* $O(n \log n)$.

Demonstração: Tal como sucede na grande maioria dos algoritmos, a fase correspondente à ordenação dos pontos é aquela que necessita de uma maior complexidade temporal. Este algoritmo não é excepção e, no pior dos casos, correspondente à situação em que os pontos se encontram alinhados em linha recta, a complexidade temporal deste algoritmo é $O(n \log n)$. ♢

Apesar deste algoritmo ser um clássico, continua a possuir vantagens óbvias relativamente aos algoritmos apresentados anteriormente. Destaca-se o facto de descartar um elevado número de pontos logo na primeira fase do algoritmo. Esta vantagem torna-se mais evidente se o conjunto de entrada tiver um elevado número de pontos.
No que se refere à fase da ordenação dos pontos, ela é bem mais simples do que a sugerida por *Graham* em seu algoritmo, uma vez que usa apenas o produto externo e não envolve cálculos com ângulos, o que seria bem mais complexo.

3.3.8 Algoritmo de Chan

O algoritmo que se segue foi inicialmente proposto por Kirkpatrick e Seidel [31], em 1986, e ao contrário do que sucede com os algoritmos que usam o método da divisão e conquista baseia-se, numa primeira fase, na *união* e só depois avança para a *conquista*. Este algoritmo de estrutura bastante complexa foi, dez anos mais tarde, apresentado por Timothy Chan [11] de uma forma bem mais simples, ficando conhecido como *Algoritmo de Chan*.

O Algoritmo de Chan combina de forma interessante dois algoritmos clássicos para a construção de invólucros convexos cuja complexidade é óptima para o pior dos casos: o *Algoritmo de Graham* e o *Algoritmo de Jarvis*. No que se refere à sua complexidade, consegue ser mais eficiente do que qualquer um dos algoritmos até aqui apresentados.
Notemos que no Algoritmo de Graham a maior parte do tempo dispendido diz respeito

à ordenação dos pontos, independentemente do tamanho do conjunto dos vértices do invólucro. Por outro lado, o Algoritmo de Jarvis torna-se mais eficiente se alguns dos vértices se encontrarem no invólucro levando, nesse caso, a uma complexidade temporal de $O(hn)$, onde h designa o número de arestas do invólucro convexo.

O Algoritmo de Chan começa por dividir um conjunto de entrada P em vários subconjuntos com o mesmo número de elementos. Se n for o número de elementos do conjunto P, então existem $r = \lceil \frac{n}{m} \rceil$ subconjuntos com m pontos, no máximo.

Observação 3.3.24 *A forma como se determina o valor de m encontra-se descrita na página* 101.

Seguidamante, recorrendo ao Algoritmo de Graham, ele contrói o invólucro convexo de cada um dos subconjuntos determinados no procedimento anterior. Por fim, usando o Algoritmo de Jarvis, é construído o invólucro convexo do conjunto inicial P. No entanto, a aplicação do Algoritmo de Jarvis não é tão directa, sendo necessário um algoritmo que permita obter as semi-rectas tangentes a um ponto e a um polígono com m lados, o que pode ser feito em $O(log\, m)$.

Obtenção das semi-rectas tangentes

Podemos obter as semi-rectas tangentes a um ponto e a um polígono da seguinte forma:

- ◇ Dados um polígono $P = \{p_0, ..., p_{m-1}\}$ com m vértices, orientados positivamente, e um ponto q que se encontra no exterior do polígono. Pretende-se obter uma semi-recta com origem em q e que é tangente ao polígono num ponto t tal que o triângulo $\Delta[q, t, p_i]$, onde p_i é um qualquer vértice do polígono, tenha orientação positiva (ver Figura 3.27).

◇ Admita-se que os vértices do polígono P se encontram armazenados numa lista, em que os índices são considerados módulo m.

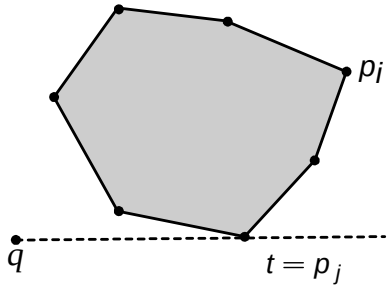


Figura 3.27: Semi-recta tangente ao polígono no ponto t.

◇ O algoritmo efectua uma análise ao vértice p_j sabendo que o vértice t procurado (ponto de tangência) se encontra entre p_j e p_{j+a-1}, entre p_j e p_{j-b+1} ou, inicialmente, $a = b = m$ (ver Figura 3.28).

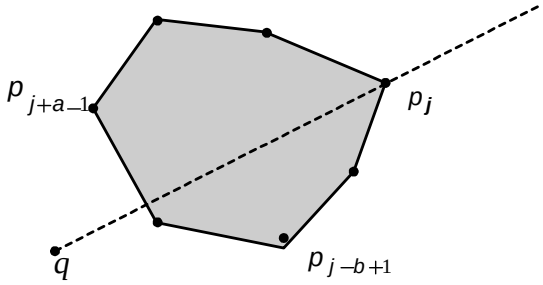


Figura 3.28: Procura do ponto de tangência t.

◇ No que se refere ao ponto p_j e à sua posição relativamente aos pontos p_{j+1} e p_{j-1}, podemos distinguir quatro situações:

- **Caso 1**: Os triângulos $\Delta[q, p_j, p_{j+1}]$ e $\Delta[q, p_j, p_{j-1}]$ têm orientação positiva. Neste caso, o algoritmo termina quando $t = p_j$.

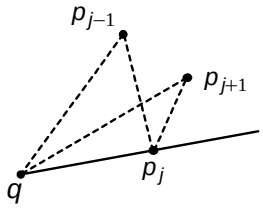


Figura 3.29: Os dois triângulos têm orientação positiva.

- **Caso 2**: O triângulo $\Delta[q, p_j, p_{j+1}]$ tem orientação positiva e o triângulo $\Delta[q, p_j, p_{j-1}]$ tem orientação negativa. Neste caso, t está entre p_{j-b+1} e p_j.

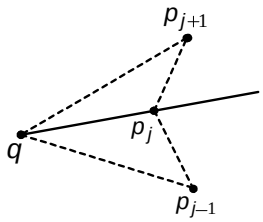


Figura 3.30: Um dos triângulos tem orientação positiva.

- **Caso 3**: O triângulo $\Delta[q, p_j, p_{j+1}]$ tem orientação negativa e o triângulo $\Delta[q, p_j, p_{j-1}]$ tem orientação positiva. Neste caso, t está entre p_j e p_{j+a-1}.

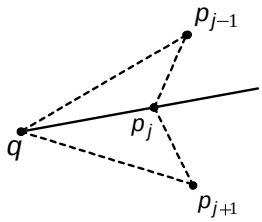


Figura 3.31: Um dos triângulos tem orientação positiva.

- **Caso 4**: Os triângulos $\Delta[q, p_j, p_{j+1}]$ e $\Delta[q, p_j, p_{j-1}]$ têm orientação negativa. Aqui, se a orientação do triângulo $\Delta[q, p_j, p_{j+a-1}]$ for positiva, t está entre p_j e p_{j+a-1}. Caso contrário, está entre p_{j-b+1} e p_j.

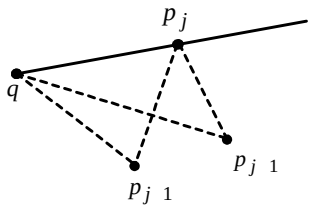


Figura 3.32: Os dois triângulos têm orientação negativa.

As instruções anteriores permitem-nos determinar algoritmicamente as tangentes pretendidas.

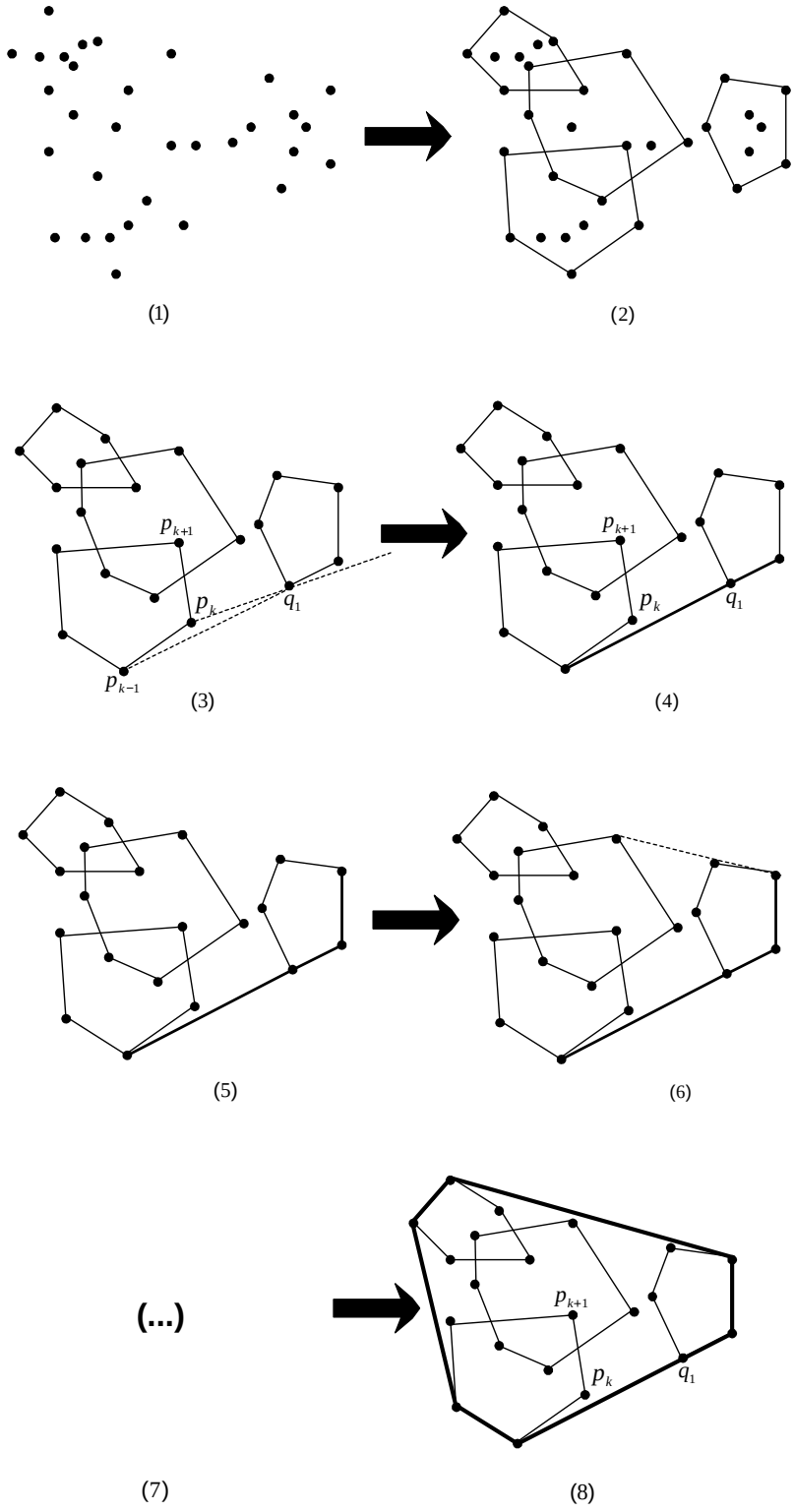


Figura 3.33: Algoritmo de Chan em acção.

A complexidade deste algoritmo é muito reduzida uma vez que em todo este processo existem, no máximo, $\log_2 m$ iterações, e como cada teste tem complexidade $O(1)$, o algoritmo corre, no pior dos casos, em $O(\log m)$.

Pseudo-código do Algoritmo de Chan (Invólucro convexo parcial)

1. Dividir S em $r = \lceil \frac{n}{m} \rceil$ subconjuntos disjuntos $S_1, ..., S_r$ com m pontos no máximo
2. Para $i = 1$ até r fazer
3. Construir o $CH(S_i)$ usando o algoritmo de Graham, armazenando os vértices numa lista ordenada
4. Seja $p_0 = (-\infty, 0)$ e $p_1 = 0$ o ponto com ordenada mínima
5. Para $k = 1$ até m fazer
6. Para $i = 1$ até r fazer
7. Determinar o ponto q_i de S_i que maximiza o ângulo $\angle p_{k-1} p_k q$
8. Fazer p_{k+1} =ponto $q \in \{q_1, ..., q_r\}$ que maximiza o ângulo $\angle p_{k-1} p_k q$
9. Se $p_{k+1} = p_1$ então retornar $\langle p_1, ..., p_k \rangle$
10. Retornar "m muito pequeno".

Teorema 3.3.25 *O Algoritmo de Chan tem complexidade temporal $O(n \log h)$.*

Demonstração: O Algoritmo de Chan divide o conjunto inicial em $r = \lceil \frac{n}{m} \rceil$ subconjuntos com o mesmo número de elementos m. O Algoritmo de Graham precisa de $O(m \, log \, m)$ para construir o invólucro convexo de cada subconjunto. Assim, na construção dos invólucros convexos dos vários subconjuntos ele necessita de $O(r \, m \, log \, m) = O(n \ log \ m)$. Seguidamente, na aplicação do Algoritmo de Jarvis aos vários subconjuntos, é necessário a construção das rectas tangentes que passam pelo ponto e pelo polígono convexo, o que leva $O(log \, m)$. O Algoritmo de Jarvis correrá em h passos, mas não podemos esquecer que temos r subconjuntos. Assim, temos uma complexidade temporal de $O(hr \, log \, m) = ((\frac{hn}{m}) log \, m)$. Combinando esta complexidade com a

complexidade correspondente à fase em que é aplicado o Algoritmo de Graham a cada subconjunto, obtemos: $O((n + \frac{hn}{m})\ log\, m)$.
A complexidade pretendida é conseguida se conseguirmos prever um valor de m tal que $m \simeq h$, obtendo-se uma complexidade temporal de $O(n\, log\, h)$.

$\diamondsuit$

A prova do resultado anterior mostrou que a eficiência deste algoritmo depende do valor de h. Assim, se conseguirmos encontrar um valor para h próximo de m conseguimos fazer com que este algoritmo tenha um bom desempenho.

Existem vários processos que nos permitem encontrar o valor de h. O mais óbvio é tentar adivinhar esse valor, fazendo $m = 1, 2, 3, ...$, o que pode levar bastante tempo. Outro dos processos usados é a aplicação de uma pesquisa binária, mas mesmo esta pesquisa pode levar $O(n\, log\, n)$ se considerarmos um valor muito alto de m.

A solução para este problema consiste em começar com um valor m muito pequeno e ir incrementando rapidamente. Como a dependência de m está no termo log, podemos chamar uma rotina com $m = h^c$, onde c é uma constante, obtendo complexidade $O(n\, log\, h)$. Para tal, basta considerar $m = 2^k$, com $k = 2^1, 2^2, ...$

Pseudo-código do Algoritmo de Chan (CH9)

1. Para $t = 1, 2, ...$fazer
2. $\quad L \leftarrow$ algoritmo invólucro convexo parcial de Chan, onde $m = h = min\{2^{2^t}, n\}$
3. $\quad$ Se $L \neq\ Incompleto$ tentar novamente e retornar a L.

Este algoritmo para cada iteração t leva $O(n \log 2^{2^t})$=$O(n\, 2^t)$. Sabemos, ainda, que este só termina quando $2^{2^t} \geq h$, o que acontece se $t = \lceil \lg \lg n \rceil$. Assim, a sua complexidade temporal será:

$$\sum_{t=1}^{\lg \lg h} n2^t = n \sum_{t=1}^{\lg \lg h} 2^t \leq n2^{1+\lg \lg h} = 2n \lg h = O(n \log h).$$

3.3.9 Algoritmo Incremental em $\mathbb{R}^2$

Os algoritmos até agora apresentados evidenciaram um sucessivo decréscimo em termos de complexidade. Alguns deles, como por exemplo o Algoritmo de Graham, possuem complexidades muito reduzidas conseguindo determinar o invólucro convexo de um conjunto de pontos em tempo óptimo de $O(n\,log\,n)$.
Embora a procura de novos algoritmos pareça de certa forma infundada, o aparecimento de novas técnicas algorítmicas e a extensão do problema da determinação do invólucro convexo ao espaço tridimensional fornecem a motivação necessária na busca de novos algoritmos. É neste sentido que surge o algoritmo que, em seguida, se apresenta. Sustentado na estratégia incremental, ele terá aplicabilidade no espaço tridimensional, como se verá mais adiante.

Neste algoritmo, o invólucro convexo é construído partindo de um conjunto previamente ordenado e a partir daí todo o processo decorre de uma forma sequencial.

A ideia base deste Algoritmo Incremental consiste em "incrementar"pontos, um de cada vez e, em cada passo, efectuar a construção do invólucro convexo dos pontos até aí incrementados. Assim, este algoritmo começa por determinar o invólucro convexo dos primeiros k pontos e, seguidamente, incrementa o próximo ponto ao invólucro anteriormente construído. Este procedimento é repetido sucessivamente até que o invólucro convexo final esteja construído.

Pseudo-código do Algoritmo Incremental (CH10)

Entrada: Um conjunto finito de pontos no plano, $S = \{p_0, ..., p_{n-1}\}$.
Saída: O invólucro convexo de S, $CH(S)$.

1. $P_2 \leftarrow CH(\{p_0, p_1, p_2\})$.
2. Para $k = 3, ..., n-1$ fazer
3. $\qquad P_k \leftarrow CH(P_{k-1} \bigcup \{p_k\})$.
4. Retornar P_{k-1}.

Seja $S = \{p_1, p_2, ..., p_n\}$ um conjunto de pontos, onde não existem três pontos colineares. Sejam $\{p_1, p_2, p_3\}$ os primeiros três pontos de S. O algoritmo irá determinar o $CH(\{p_1, p_2, p_3\})$ que será um triângulo.
Seja $Q = P_{k-1}$ e $p = p_k$, na construção do invólucro convexo podemos distinguir as seguintes situações:

- $\diamond$ $p \in Q$. É evidente que se $p \in Q$, então p pode ser eliminado. O mesmo irá acontecer caso p se encontre na fronteira de Q. A verificação de que $p \in Q$ pode ser feita recorrendo à primitiva *LeftOn*, ou seja, $p \in Q$ se e só se p se encontrar à esquerda ou sobre uma aresta orientada de Q. Esta verificação pode ser efectuada em $O(n)$ tempo complexidade.

- $\diamond$ $p \notin Q$. Caso se venha a verificar que o ponto p não obedece à primitiva *LeftOn*, então $p \notin Q$. Nesse caso, teremos que determinar o $CH(Q \bigcup \{p\})$, o que pode ser feito de forma simples traçando duas semi-rectas tangentes com origem no ponto p e que são tangentes ao polígono, tal como se pode observar na Figura 3.34.

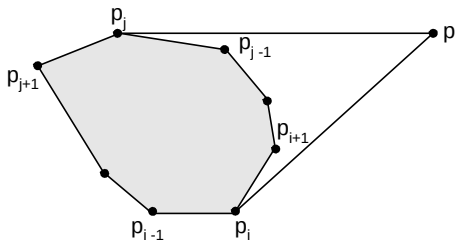


Figura 3.34: Rectas que passam pelo ponto p e que são tangentes ao polígono Q nos pontos p_i e p_j.

Recorrendo à primitiva *LefOn* verifica-se que o ponto de tangência p_i se encontra à esquerda de $[p_{i-1}, p_i]$ e à direita de $[p_i, p_{i+1}]$. O mesmo raciocínio é usado para encontrar o ponto de tangência p_j. Notemos que estes dois pontos de tangência, p_i e p_j, são encontrados recorrendo a sucessivos testes da primitiva *LeftOn*, para verificar se $p \in$ $\in Q$. Assim, para $p \notin Q$ esta verificação pode ser feita num tempo complexidade de $O(n)$.

Pseudo-código do algoritmo pontos de tangência

Entrada: Um polígono convexo Q com m vértices e um ponto p tal que $p \notin Q$.

Saída: As duas semi-rectas com origem no ponto p e são tangentes ao polígono Q.

1. Para $i = 1, ..., m$ fazer
2. Se LeftOn $(p_{i-1}, p_i, p) \neq$ LeftOn (p_i, p_{i+1}, p)
3. Então p_i é ponto de tangência.

Considerando o polígono Q, os dois pontos de tangência e o ponto p, o novo invólucro convexo fica definido por: $p_1, p_2, ..., p_i, p, p_j, p_{j+1}, ..., p_n$.

Caso os pontos extremos do invólucro estejam representados por uma lista circular duplamente ligada [2], então, a actualização do invólucro convexo poderá ser feita recorrendo a um conjunto de remoções e uma inserção.

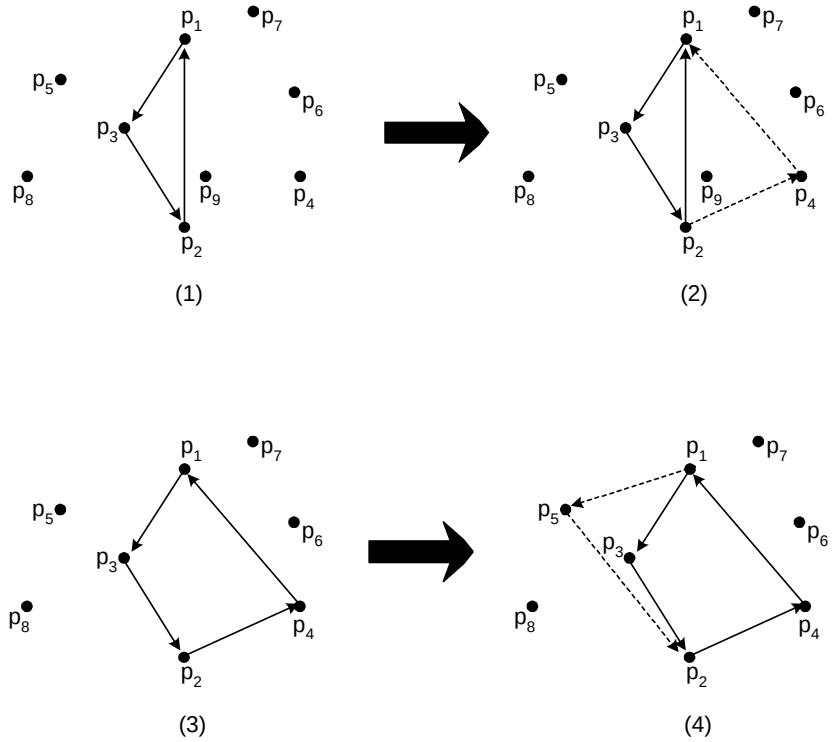


Figura 3.35: Algoritmo Incremental em acção.

[2]É uma lista circular onde cada elemento, ou nó, é composto por uma variável que guarda informação e ponteiros (referências a endereços de memória) que permitem a ligação entre os vários nós desta lista.

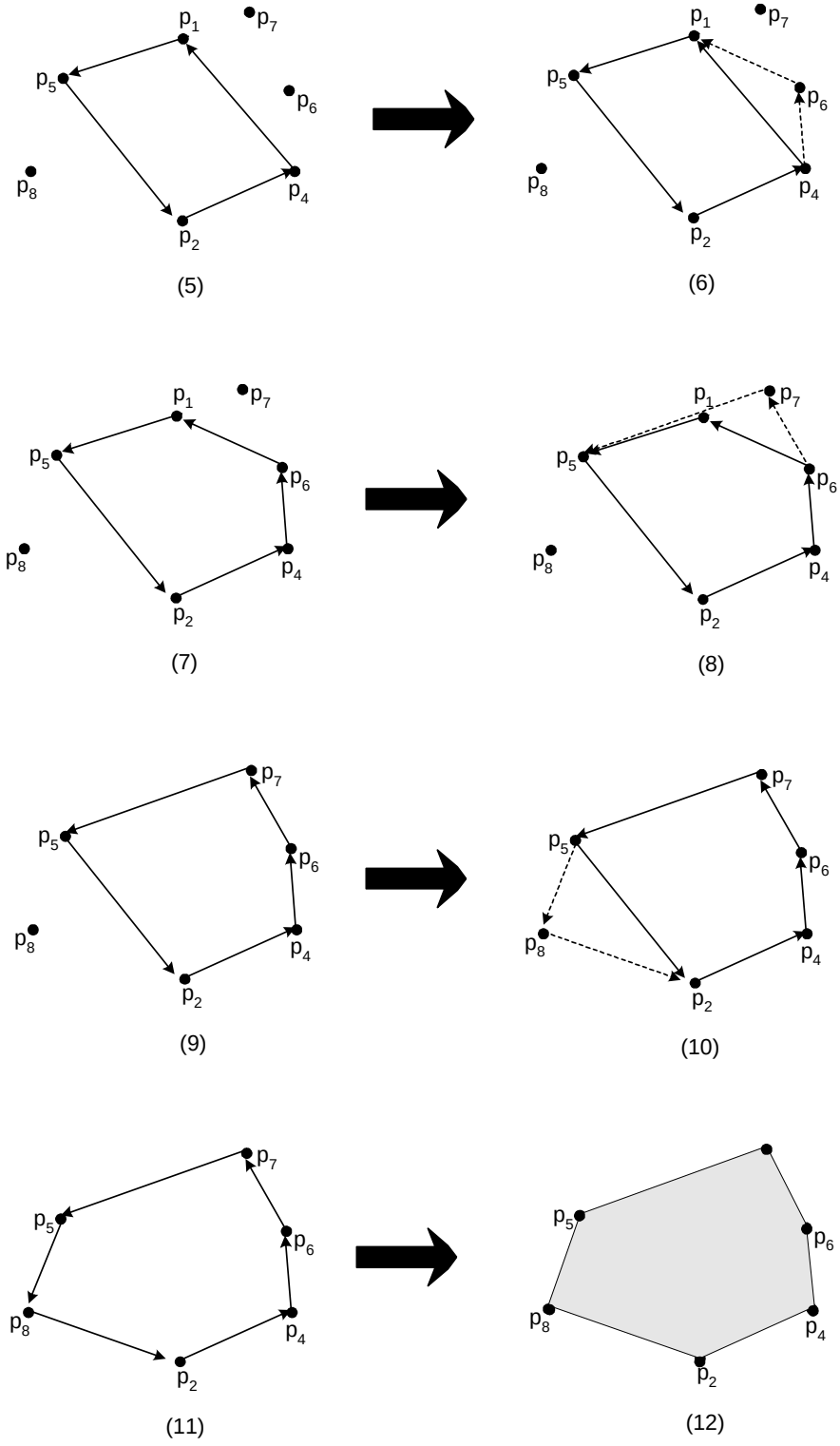


Figura 3.36: Algoritmo Incremental em acção.

Teorema 3.3.26 *O Algoritmo Incremental tem complexidade temporal* $O(n^2)$.

Demonstração: A verificação de que $p \in Q$ pode ser feita em $O(n)$. Caso $p \notin Q$, encontrar os dois pontos de tangência requer tempo $O(n)$. As actualizações (remoções e inserções) da lista circular onde os pontos extremos se encontram armazenados pode também ser feita em tempo $O(n)$, uma vez que o número total de remoções durante a execução do algoritmo é menor que o número total de inserções que é igual a n. Assim, teremos uma complexidade de $O(n^2)$. $\diamondsuit$

A complexidade do Algoritmo Incremental pode, no entanto, ser minorada. Se adaptarmos este algoritmo a um algoritmo probabilístico, a sua complexidade ficará reduzida para $O(n \log n)$.

3.4 Limite inferior no problema da construção de invólucros convexos

No decorrer deste capítulo, observámos que o problema da determinação do invólucro convexo de um conjunto finito de n pontos no plano pode ser resolvido algoritmicamente com complexidade $O(n\, log\, n)$, no melhor dos casos. Nesta secção, iremos mostrar que os algoritmos que resolvem este problema com complexidade $O(n\, log\, n)$ são os mais eficientes, isto é, $\Omega(n \log n)$ é o limite inferior para o problema do invólucro convexo no plano. Para tal, iremos usar a redução entre problemas.

Um Problema A pode ser reduzido para um Problema B sempre que o algoritmo que é utilizado para resolver o Problema B pode também ser usado para resolver o Problema A, adicionando-lhe um pequeno " esforço"em termos de complexidade. Assim, se existir um algoritmo que resolva o Problema B rapidamente, é certo que o Problema A será rapidamente solucionado. Suponhamos que um Problema A tem limite inferior $\Omega(n^3)$ e que o Problema A pode ser reduzido a um problema B, isto é, adicionando um pequeno " esforço"ao algoritmo que resolve o Problema B podemos resolver o Problema A. Se

$O(n)$ traduzir o esforço que se adiciona ao algoritmo que resolve o Problema B e se recorrermos apenas uma vez ao algoritmo que resolve o Problema B, podemos concluir que como $\Omega(n^3)$ é o limite inferior do Problema A, então $\Omega(n^2)$ será o limite inferior do Problema B.

Como já referimos, a determinação de limites inferiores não triviais para problemas pode tornar-se num processo complicado. No entanto, já se conhecem limites inferiores para alguns problemas importantes, como por exemplo, o Problema de Ordenação de n pontos no plano. Ficou provado no Teorema 3.2.7 (Secção 3.2.2), que o limite inferior no Problema de Ordenação de n pontos no plano é $\Omega(n\ log\ n)$. O que faremos, em seguida, é uma redução do Problema de Ordenação (Problema A) para o Problema da determinação do Invólucro Convexo de um conjunto de n pontos no plano. Seguidamente, iremos apresentar uma proposta de redução de problemas sugerida por Shamos [43].

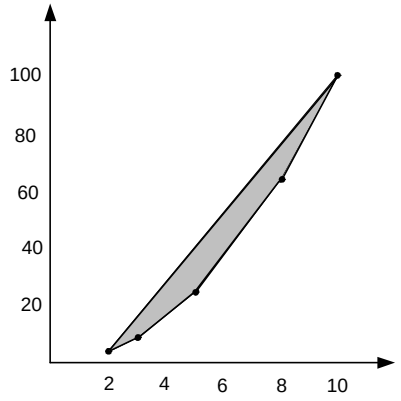


Figura 3.37: Parábola obtida da ordenação dos pontos $2, 3, 5, 8$ e 10.

Consideremos uma lista não ordenada de números $\{x_1, ..., x_n\}$, $x_i \geq 0$ $(i = 1, ..., n)$ e um algoritmo CH que constrói o invólucro convexo de um conjunto de n pontos no plano. Seja $T(n)$ a função complexidade do algoritmo CH. A ordenação dos n números pode ser feita recorrendo ao algoritmo CH em tempo $T(n) + O(n)$, onde $O(n)$ representa o tempo necessário à transformação da instância do Problema de Ordenação para uma instância do Problema da determinação do Invólucro Convexo e transformar uma solução do Problema da determinação do Invólucro Convexo numa solução do Problema de Ordenação.

Considere-se o conjunto de pontos do plano constituído pelos pontos da forma (x_i, x_i^2) $(i = 1, ..., n)$ (ver Figura 3.37). Recorrendo ao algoritmo CH é possível determinar o invólucro convexo do conjunto de pontos da Figura 3.37. Depois, é procurado no invólucro convexo o ponto $p := (x_k, x_{k+1})$ de ordenada miníma, o que é feito em $O(n)$. Percorrendo a fronteira do invólucro convexo, em sentido anti-horário, a partir do ponto p, obtemos uma lista de números, onde estes aparecem ordenados por ordem crescente.

A redução entre problemas que acabamos de efectuar permite-nos concluir que o algoritmo CH pode ser utilizado para ordenar n números em tempo $O(T(n))$, onde $T(n) = \Omega(n\ log\ n)$. Desta forma, fica evidente que $O(n\ log\ n)$ é o melhor que se consegue ter quando se pretende construir o invólucro convexo de um conjunto de pontos no plano. Este é também um resultado válido em $\mathbb{R}^3$.

3.5 Dois algoritmos para a construção de invólucros convexos no espaço

O objectivo central desta secção é apresentar, a título conclusivo, dois dos algoritmos já estudados que determinam o invólucro convexo de um conjunto finito de pontos no plano e que têm também aplicabilidade no espaço tridimensional.

Tal como era de se esperar, os algoritmos que permitem a construção de invólucros convexos no espaço tridimensional são bem mais complexos do que a generalidade dos algoritmos com aplicabilidade no plano. Por esse motivo, nesta secção, serão apenas abordados dois: o *Algoritmo Gift Wrapping* (ou "embrulho para presente") e o *Algoritmo Incremental*. Estes algoritmos serão abordados de uma forma que se considera superficial, uma vez que um estudo mais detalhado não se enquadra no objectivo central desta dissertação.

3.5.1 Organização de dados

Quando, em $\mathbb{R}^2$, resolvemos algoritmicamente o problema da determinação do invólucro convexo de um conjunto de pontos, sabemos que a solução produzida pelo algoritmo será um polígono convexo, que é uma estrutura de simples representação. No entanto, quando estendemos o problema a $\mathbb{R}^3$ a solução já não é tão simples, ficando condicionada à forma como os dados se encontram representados. É claro, que representações diferentes conduzem-nos a diferentes complexidades, condicionando o mau ou bom desempenho de um algoritmo. Assim, passaremos ao estudo de formas que nos permitem representar o invólucro convexo no espaço tridimensional.

A representação de um poliedro pode ser feita de forma simples, recorrendo a uma lista ordenada de vértices. Nesse caso, o poliedro fica caracterizado através de uma lista de faces (polígonos), que constituem a sua superfície. No entanto, esta representação é muito limitativa uma vez que só contém informação acerca da adjacência entre os vértices de uma mesma face.

3.5.1.1 Organização de dados em poliedros simpliciais

Definição 3.5.1 *Um poliedro simplicial é um poliedro limitado por faces triangulares.*

Para representar um poliedro simplicial são utilizadas listas duplamente ligadas no armazenamento independente dos vértices, das arestas e das faces. A estrutura do nó responsável pelo vértice possuirá as respectivas coordenadas. A estrutura do nó responsável pela aresta irá conter apontadores para os dois vértices extremos e para as duas faces que lhe são incidentes. Finalmente, o nó de uma face irá conter apontadores para os três vértices que formam a face e três apontadores para as três arestas.
Notemos que esta estrutura só permite a representação de poliedros simpliciais devido à forma como se define o tipo de dados para as faces (três vértices e três arestas).

3.5.1.2 Organização de dados usando a estrutura *winged-edge*

A estrutura de dados *winged-edge* (ou "aresta-alada") apresenta vantagens relativamente às estruturas anteriores, uma vez que representa um poliedro recorrendo a um grafo planar, o que nos garante todo um conjunto de informações pertinentes, nomeadamente, quanto às possíveis relações de adjacência e de incidência. Na Figura 3.38 podemos observar uma ilustração deste tipo de estrutura.

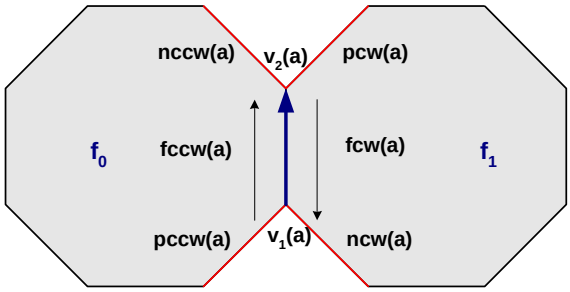


Figura 3.38: Estrutura de dados *winged-edge*.

A estrutura de dados em *winged-edge* mantém as listas de vértices, arestas e faces, onde:

- ◇ **vértice**: a estrutura que armazena cada vértice v contém as coordenadas (x, y, z) do vértice juntamente com o apontador para uma qualquer aresta $av(v)$ incidente no vértice v.
- ◇ **face**: cada face f contém um apontador para uma qualquer aresta $af(f)$ contida na fronteira de f.
- ◇ **aresta**: a estrutura responsável pelas arestas possui 8 apontadores por cada aresta a:
  - dois apontadores para os vértices $v_1(a)$ e $v_2(a)$, que são os extremos da aresta a. A ordem pela qual surgem estes vértices dá-nos a orientação da aresta.
  - dois apontadores para as duas faces fccw(a) (*counterclockwise*) e fcw(a) (*clockwise*) incidentes em a, onde fccw(a) designa a face com orientação positiva e fcw(a) a face com orientação negativa.

- quatro apontadores para as arestas adjacentes a a ("asas"): pccw(a) (*previus counterclockwise*), nccw(a) (*next counterclockwise*), pcw(a), e ncw(a), ou seja, as arestas que precedem e sucedem a fccw(a) e a few(a), respectivamente.

Na Figura 3.39 podemos observar um cubo e sua respectiva representação num grafo planar.

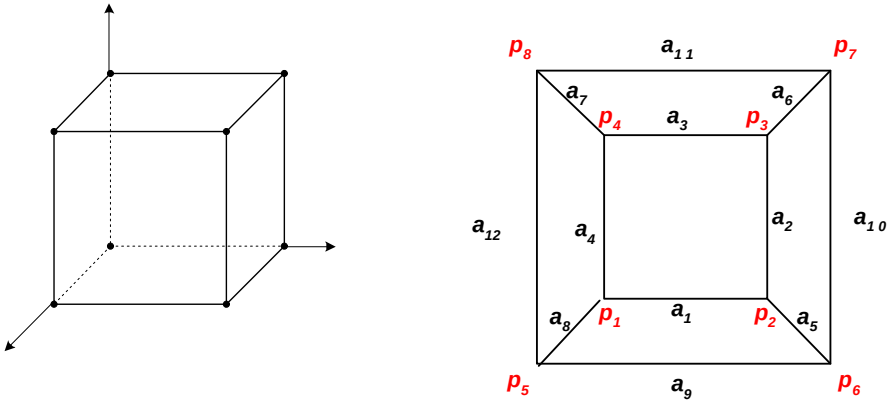


Figura 3.39: Cubo e respectivo grafo planar.

A estrutura *winged-edge*, correspondente ao cubo da Figura 3.39, encontra-se representada nas tabelas seguintes:

vértice	(x, y, z)	**av**
p_1	$(1, 0, 0)$	a_1
p_2	$(1, 1, 0)$	a_2
p_3	$(1, 1, 1)$	a_3
p_4	$(1, 0, 1)$	a_4
p_5	$(0, 0, 0)$	a_9
p_6	$(0, 1, 0)$	a_{10}
p_7	$(0, 1, 1)$	a_{11}
p_8	$(0, 0, 1)$	a_{12}

face	**af**
f_1	a_1
f_2	a_2
f_3	a_{10}
f_4	a_{12}
f_5	a_1
f_6	a_6

arestas	v_1	v_2	**fccw**	**fcw**	**pccw**	**nccw**	**pcw**	**ncw**
a_1	p_1	p_2	f_1	f_5	a_4	a_2	a_5	a_8
a_2	p_2	p_3	f_1	f_2	a_1	a_3	a_6	a_5
a_3	p_3	p_4	f_1	f_6	a_2	a_4	a_7	a_6
a_4	p_4	p_1	f_1	f_4	a_3	a_1	a_8	a_7
a_5	p_2	p_6	f_2	f_5	a_2	a_{10}	a_9	a_1
a_6	p_3	p_7	f_6	f_2	a_3	a_{11}	a_{10}	a_2
a_7	p_4	p_8	f_4	f_6	a_4	a_{12}	a_{11}	a_3
a_8	p_1	p_5	f_5	f_4	a_1	a_9	a_{12}	a_4
a_9	p_6	p_5	f_3	f_5	a_{10}	a_{12}	a_8	a_5
a_{10}	p_6	p_7	f_2	f_3	a_5	a_6	a_{11}	a_9
a_{11}	p_7	p_8	f_6	f_3	a_6	a_7	a_{12}	a_{10}
a_{12}	p_5	p_8	f_3	f_4	a_9	a_{11}	a_7	a_8

3.5.2 Teste de orientação

O teste de orientação que, em seguida, se descreve é uma generalização do teste de orientação que é feito no plano. Sejam p_0, p_1 e p_2 três pontos não colineares que definem um triângulo e π um plano tal que $p_0, p_1, p_2 \in \pi$. Sejam $\vec{a} := (x_a,\ y_a,\ z_a)$ e $\vec{b} := (x_b,\ y_b\ ,\ z_b)$, onde $\vec{a} = (p_1 - p_0)$ e $\vec{b} = (p_2 - p_1)$. Então, podemos encontrar um vector $\vec{v}$ normal (ou ortogonal) a π calculando o produto vectorial $\vec{a} \times \vec{b}$:

$$\vec{a} \times \vec{b} = \det \begin{pmatrix} y_a & z_a \\ y_b & z_b \end{pmatrix} \overrightarrow{i} + \det \begin{pmatrix} z_a & x_a \\ z_b & x_b \end{pmatrix} \overrightarrow{j} + \det \begin{pmatrix} x_a & x_b \\ y_a & y_b \end{pmatrix} \overrightarrow{k}$$

A verificação de que o produto vectorial $\vec{a} \times \vec{b}$ é perpendicular ao plano π é feita mostrando que $\langle \vec{a}, (\vec{a} \times \vec{b}) \rangle = 0$ e $\langle \vec{b}, (\vec{a} \times \vec{b}) \rangle = 0$.

Se observarmos a Figura 3.40 e se considerarmos o triângulo $\Delta[O, \vec{a}, \vec{b}]$, verificamos que este possui orientação positiva.

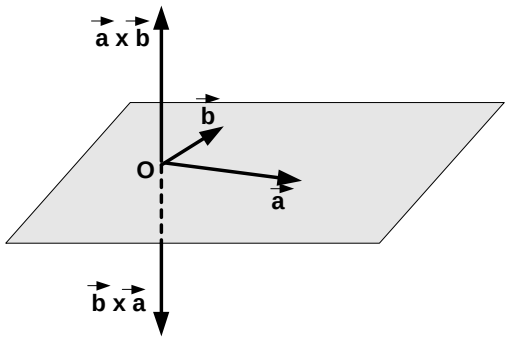


Figura 3.40: Produto vectorial $\vec{a} \times \vec{b}$

Um plano determinado por um triângulo que possua uma orientação divide o espaço em dois semi-espaços. Destes, um encontra-se voltado para o vector normal ao triângulo e designa-se por semi-espaço positivo do triângulo, uma vez que dos pontos do semi-espaço conseguimos ver o triângulo orientado positivamente. O outro semi-espaço chama-se semi-espaço negativo.

Considerem-se, em $\mathbb{R}^3$, p_0, p_1, p_2 e p_3 e um plano π gerado pelos vectores $\vec{a} := (p_1 - p_0)$ e $\vec{b} := (p_2 - p_0)$. Se pretendermos saber de que lado do plano π é que o ponto p_3 se encontra, basta considerar uma orientação de π dada pelo triângulo $\Delta[p_0, p_1, p_2]$ e verificar qual o sinal do produto escalar entre $\vec{c} := (p_3 - p_0)$ e o vector $(\vec{a} \times \vec{b})$. Se $\langle \vec{c}, (\vec{a} \times \vec{b}) \rangle > 0$, então p_3 estará no semi-espaço positivo do triângulo $\Delta[p_0, p_1, p_2]$. Se, pelo contrário, $\langle \vec{c}, (\vec{a} \times \vec{b}) \rangle < 0$ então p_3 estará no semi-espaço negativo. Caso o resultado deste produto seja igual a zero, p_3 estará no plano π.

O teste de orientação que acabamos de descrever é o mesmo que foi utilizado na execução dos algoritmos em $\mathbb{R}^2$ e é, como já foi referido, conhecido como primitiva *Left*. Este teste possui a vantagem de poder ser aplicado a um espaço qualquer que seja a sua dimensão.

3.5.3 Linearidade entre o número de arestas e faces de um poliedro

Seguidamente, encontraremos limites superiores para as arestas e faces de um poliedro. Sejam v, a e f o número de vértices, arestas e faces de um poliedro, respectivamente. Considere-se, sem perda de generalidade, que as faces de um poliedro são triangulares, o que é possível dado que caso as faces não sejam triangulares é sempre possível efectuar uma triangularização. Assim, o número de arestas a de um poliedro será $3f/2$. Consequentemente, $2a = 3f$. Aplicando a Fórmula de Euler (ver Teorema 2.4.3) tem-se que:

$$\begin{aligned} v - a + 2a/3 &= 2 \\ v - 2 &= a - 2a/3 \\ v - 2 &= a/3 \\ a &= 3v - 6 < 3v = 3n \end{aligned}$$

Como $a = 3v-6 < 3v = 3n$ teremos para as arestas complexidade $O(n)$. Relativamente às faces teremos:

$$\begin{aligned} v - 3f/2 + f &= 2 \\ v - 2 &= f/2 \\ v - 2 &= a/3 \\ f &= 2v - 4 < 2v = 2n \end{aligned}$$

Teorema 3.5.2 *Num poliedro com $n = v$ vértices, a arestas e f faces, $v - a + f = 2$, $a = O(n)$ e $f = O(n)$.*

3.5.4 Algoritmo Gift-Wrapping em $\mathbb{R}^3$

A técnica Gift-Wrapping ("embrulho-para-presente") apresentada por Chand e Kapur [12], constitui uma generalização do Algoritmo de Jarvis, apresentado na secção anterior.

O Algoritmo Gift-Wrapping desenvolve-se de modo muito idêntico no plano e no espaço. No entanto, a sua implementação no espaço é mais complexa.

O algoritmo Gift-Wrapping inicia-se de forma muito semelhante ao Algoritmo de Jarvis. Deste modo, começa-se por seleccionar um ponto extremo $p_0 \in S$ que pode, por exemplo, corresponder ao ponto com menor valor de $z-coordenada$. Caso exista um empate deve-se escolher o ponto com menor valor de ordenada. Se, mesmo assim, o empate continuar opta-se pelo valor mais baixo de abcissa. Seguidamente, considera-se uma recta r que contém o ponto p_0 e é paralela ao eixo das abcissas e um semi-plano horizontal π, orientado positivamente relativamente ao eixo das ordenadas, que contém p_0. Posteriormente, o semi-plano π é rotacionado em torno da recta r até intersectar um novo ponto do conjunto S, seja p_1. Considerando a aresta definida pelos dois pontos extremos anteriormente encontrados, p_0 e p_1, rotaciona-se novamente π em torno desta aresta até que este encontre um novo ponto de S, seja p_2. O triângulo formado pelos pontos p_0, p_1 e p_2 constitui a primeira face do invólucro convexo de S, como se pode observar na Figura 3.41.

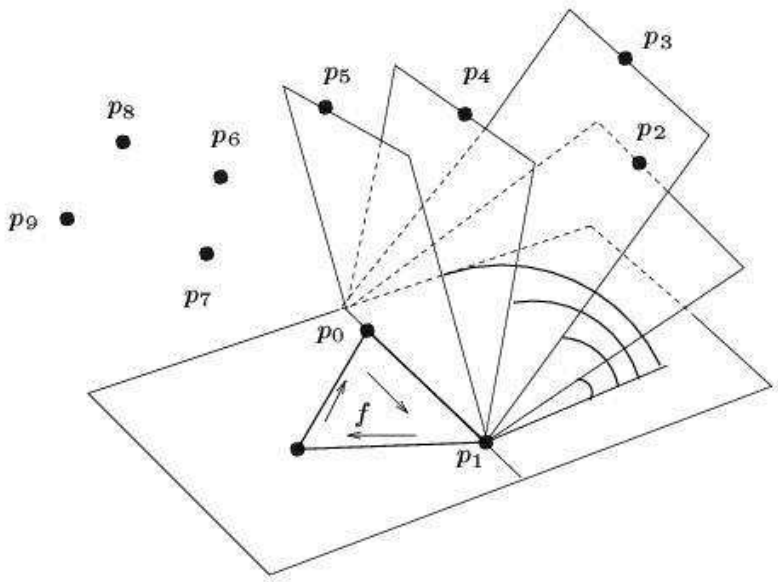


Figura 3.41: Passo central do Algoritmo Gift Wrapping em $\mathbb{R}^3$.

O algoritmo segue aplicando sucessivamente o procedimento anteriormente descrito na obtenção de novas faces, ou seja, utilizando arestas das faces já conhecidas na obtenção de novas faces. Deste modo, se f for uma face encontrada num dos passos anteriores deste algoritmo e se a for uma aresta de f cuja face que lhe é adjacente ainda não foi encontrada, o plano que contém a face f irá ser rotacionado em torno de a até que um novo ponto p do conjunto seja encontrado. Neste caso, o invólucro convexo será $CH(a \cup \{p\})$. Para cada face encontrada e para cada aresta que lhe é adjacente, o algoritmo encontra a face adjacente. Assim, no passo seguinte, ele irá analisar uma nova face e para essa face ele irá verificar se as arestas que lhe são adjacentes pertencem ou não a uma face já determinada. Caso isto se venha a verificar, as duas faces incidentes nesta aresta já estão encontradas fazendo com que a aresta não precise mais de ser examinada. Se, pelo contrário, existir uma face incidente nessa aresta que ainda não foi determinada, a aresta é assumida como aresta livre, ou seja, será analisada novamente pelo algoritmo. O algoritmo termina assim que todas as arestas sejam encontradas.

Pseudo-código do Algoritmo Gift Wrapping 3D (CH11)

Entrada: Um conjunto finito de pontos $S = \{p_0, p_1, ..., p_{n-1}\}$, em $\mathbb{R}^3$.

Saída: O invólucro convexo de S, $CH(S)$.

1. $Q \leftarrow \emptyset$.
2. $f \leftarrow$ uma face inicial de CH(S).
3. Insira f na fila Q e na estrutura *winged-edge* T.
4. Enquanto $Q \neq \emptyset$ fazer.
5. $\quad f \leftarrow$ primeira face na fila Q.
6. $\quad$ Para cada aresta livre a de f fazer
7. $\quad\quad f' \leftarrow$ face do invólucro convexo que compartilha a com f.
8. $\quad\quad$ Insira f' na fila Q.
9. $\quad\quad$ Insira f' na estrutura winged-edge T, ligando-a com as faces já geradas que lhe são adjacentes e determinando as arestas livres.
10. Retornar T.

Vejamos o que sucede, em termos de complexidade, ao Algoritmo *Gift Wrapping* em $\mathbb{R}^3$.

Teorema 3.5.3 *O Algoritmo Gift Wrapping constrói o invólucro convexo de um conjunto com n pontos, em* $\mathbb{R}^3$*, com complexidade temporal* $O(nh)$*, onde h designa o número de arestas do invólucro convexo.*

Demonstração: O procedimento central neste algoritmo consiste na fase correspondente ao "embrulho-para-presente", que decorre no passo 7 (ver pseudo-código de CH11). Esta fase é realizada com complexidade $O(n)$. Como o invólucro convexo possui h arestas, temos que este passo irá necessitar de complexidade $O(nh)$ durante a execução de todo o algoritmo.
Para cada face encontrada é necessário ver se as arestas desta face já foram criadas, o que leva tempo $O(n)$. Como cada aresta é gerada duas vezes, temos que o tempo total gasto neste teste durante todo o algoritmo será $O(nh)$. Da Fórmula de Euler conclui-se que $O(nh) = O(n^2)$. ♢

Como acabamos de verificar, a complexidade temporal do Algoritmo Gift Wrapping depende do número de arestas h do invólucro convexo. Estamos, portanto, perante um algoritmo do tipo *output-sensitive*.

3.5.5 Algoritmo Incremental

O Algoritmo Incremental para determinar o invólucro convexo, em $\mathbb{R}^3$, é estruturalmente idêntico ao Algoritmo Incremental estudado na Subsecção 3.3.9. Deste modo, ele examina os pontos do conjunto um de cada vez e, em cada passo, constrói o invólucro convexo dos pontos já examinados. O algoritmo tem continuidade com a construção progressiva do invólucro convexo recorrendo ao invólucro convexo anteriormente determinado e ao ponto que está a ser examinado nesse momento.

Note-se que o invólucro convexo dos primeiros 4 pontos corresponde a um tetraedro. Seja $Q = P_{k-1}$ e $p = P_k$. Tal como acontece com este algoritmo quando aplicado no plano, quando se pretende determinar o $CH(Q \cup \{p\})$, podem ocorrer as situações seguintes:

- $\diamond$ $p \in Q$. Neste caso, o ponto p pode ser eliminado e $CH(Q \cup p) = Q$. A verificação de que $p \in Q$ pode ser feita recorrendo ao teste de orientação apresentado na Subsecção 3.5.2, cuja complexidade é $O(n)$. Segundo este teste, $p \in Q$ se a partir de p é possível ver as faces de Q orientadas negativamente.

- $\diamond$ $p \notin Q$. Se existir um ponto p tal que a partir dele se vê uma face f de Q, orientada positivamente, então $p \notin Q$. A forma como é determinado o $CH(Q \cup p)$ é semelhante ao caso bidimensional só que desta vez temos de encontrar planos tangentes ao ponto p. No final tem-se um cone limitado por faces triangulares.

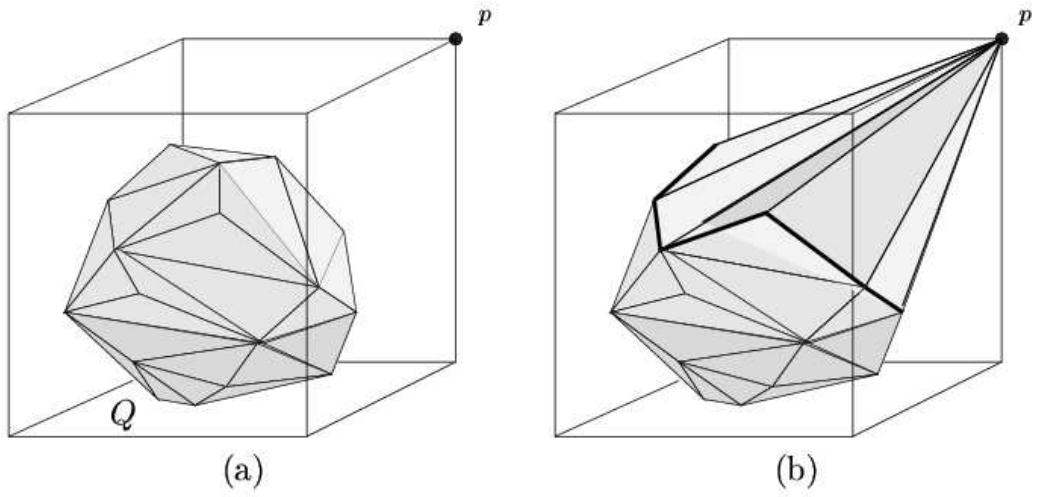


Figura 3.42: (a) poliedro antes do ponto p ser analisado pelo algoritmo (b) poliedro depois do ponto p ser analisado pelo algoritmo.

Como era esperado o pseudo-código do Algoritmo Incremental no espaço é semelhante ao algoritmo correspondente no plano.

Pseudo-código do Algoritmo Incremental 3D (CH12)

Entrada: Um conjunto finito de pontos $S = \{p_1, p_1, ..., p_n\}$, em $\mathbb{R}^3$.

Saída: O invólucro convexo de S, $CH(S)$.

1. $P_3 \leftarrow$ tetraedro formado pelos pontos p_1, p_2, p_3, p_4
2. Para $k = 45, ..., n$ fazer
3. Para cada face f de P_{k-1} fazer
4. Calcular o volume do tetraedro determinado por f e p_k.
5. Se sinal do volume for > 0 então marcar f como visível.
6. Se nenhuma face é visível por p_k
7. $P_k \leftarrow P_{k-1}$;
8. Senão
9. Para cada aresta a na fronteira das faces visíveis fazer
10. Construa a face determinada por a e P_k.
11. Para cada face visível f fazer
12. Remova f de P_{k-1}.
13. Retornar P_{k-1}.

Teorema 3.5.4 *O Algoritmo Incremental constrói o invólucro convexo de um conjunto de n pontos, em* $\mathbb{R}^3$*, com complexidade temporal* $O(n^2)$.

Demonstração: Observando o pseudo-código deste algoritmo e tendo por base a fórmula de Euler verificamos que $f = O(n)$ e $a = O(n)$, onde n é o número de vértices do polítopo. Logo, cada ciclo 3., 9. e 11. terá complexidade $O(n)$. Como cada um destes ciclos é percorrido $O(n)$ vezes, temos uma complexidade temporal $O(n^2)$. $\diamondsuit$

Capítulo 4

Conclusões e considerações finais

A presente dissertação iniciou-se com a apresentação de um estudo breve sobre conjuntos convexos onde se apresentaram resultados, nomeadamente, quanto à sua separação e representação. Este primeiro capítulo contribuiu, sem dúvida, para analisar com maior detalhe os algoritmos que resolvem o problema da determinação de invólucros convexos.

O problema da determinação do invólucro convexo de um conjunto de pontos tem sido objecto de um intensivo estudo nos últimos anos. Desde os algoritmos clássicos, como por exemplo o Algoritmo de Jarvis [29], até àqueles com desenvolvimento mais recente, como o Algoritmo de Chan [11], assistimos à implementação de novas e diversificadas técnicas que influenciam, claramente, o desempenho de cada algoritmo.

Nesta dissertação, apresentámos vários algoritmos que determinam invólucros convexos. Destes, dois utilizam o processo *força bruta* na sua execução: o Algoritmo Pontos não Extremos e o Algoritmo Arestas Extremas. No entanto, este processo revela pouca eficiência fazendo com que estes algoritmos possuam complexidades muito elevadas. No Algoritmo de Jarvis, no Algoritmo de Graham e no Algoritmo Incremental, a utilização do método *iterativo* mostra-se mais eficiente. Relativamente ao Algoritmo de Jarvis, cuja complexidade é, no pior caso, $O(nh) = O(n^2)$ (onde h designa o número de arestas do invólucro), observamos que esta pode ser reduzida para uma complexidade óptima quando se considera $h = O(\log\, n)$. No que se refere àquele que foi o primeiro algoritmo

a possuir complexidade óptima de $O(n \log n)$, o Algoritmo de Graham, verificámos que ele dispende um maior esforço na fase correspondente à ordenação dos pontos do conjunto de entrada. Assim, quando os pontos do conjunto já se encontram ordenados, este algoritmo torna-se mais eficiente e corre em tempo linear.

O Algoritmo Incremental também mostrou alguma eficiência, mas a sua complexidade pode ser óptima se a este algoritmo adaptarmos um algoritmo probabilístico. Este é um procedimento válido nos espaços bidimensional e tridimensional.

O estudo do Algoritmo da Cadeia Poligonal Monótona tornou-se interessante, uma vez que neste algoritmo é utilizado o método da cadeia poligonal monótona proposto por Lee e Preparata. Quando comparado com o Algoritmo de Graham, este algoritmo apresenta a vantagem de, na fase da ordenação, utilizar uma ordenação lexicográfica linear que é mais eficiente que a ordenação por ângulo polar presente no Algoritmo de Graham.

Nos algoritmos Quickhull e Mergehull assistimos à utilização do método da *divisão e conquista*. Este método é uma mais valia nestes algoritmos permitindo que, na generalidade dos casos, o invólucro convexo seja construído em $O(n \log n)$.

Com semelhanças evidentes com os algoritmos que têm por base a divisão e conquista, encontramos o Algoritmo proposto por Akl e Toussaint [2]. A apresentação deste algoritmo tornou-se interessante dado que ele possui uma vantagem óbvia relativamente aos algoritmos anteriores pois pode eliminar, numa primeira fase, um elevado número de pontos, reduzindo o problema inicial a um problema mais simples. Neste algoritmo, a forma como são ordenados os pontos do conjunto contribui também para o seu bom desempenho.

Um dos algoritmos com desenvolvimento muito recente é o Algoritmo de Chan que, para além de ser óptimo no espaço bidimensional, tem aplicabilidade no espaço de dimensão três.

Na fase final desta dissertação apresentámos a implementação de dois dos algoritmos estudados no plano, no espaço tridimensional: o Algoritmo Gift-Wrapping e o Algoritmo Incremental. Estes algoritmos evidenciaram muitas semelhanças na sua execução no espaço e no plano, mesmo em termos de complexidade.

Os algoritmos apresentados ao longo deste trabalho foram seleccionados de acordo com a sua relevância histórica, no âmbito dos invólucros convexos, e tendo em conta um conjunto diversificado de técnicas. Assim, esta dissertação revela-se como um bom ponto de partida para trabalhos futuros relacionados, essencialmente, com invólucros convexos. É o caso da construção dinâmica de invólucros convexos que, apesar de ter grande utilidade prática, não foi abordada mas poderia marcar uma linha de trabalho nesta temática.

Bibliografia

[1] A. V. Aho, J. E. Hopcroft e J. D. Ulman, *Design and Analysis of Computer Algorithms*, Addison-Wesley, (1976).

[2] S. G. Akl e G. T. Toussaint, *A Fast Convex Hull Algoritm*, School of Computet Science, McGill University, Montreal, Quebec, Canada, (1978).

[3] A. M. Andrew, *Another efficient algorithm for convex hull in two dimensions*, Info. Proc. Lett. 9, pp. 216-219, (1979).

[4] C. B. Barber, D. P. Dobkin e H. Huhdanpaa, *The Quickhull Algorithm for Convex Hulls*, (1995).

[5] L. M. Barbosa e D. D. Salvetti, *Algoritmos*, Makrow Books, (1998).

[6] M. de Berg, M. van Kreveld, M. Overmars e O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, (1997).

[7] A. Bykat, *Convex hull of a finite set of points in two dimensions*, Info. Proc. Lett. 7, pp. 296-298,(1978).

[8] G. S. Brodal e R. Jacob, *Dynamic Planar Convex Hull*, Department of Computer Science, University of Aarhus.

[9] V. A. Bushenkov, *An iteration method of constructing orthogonal projections of convex polyhedral sets*, Comput. Maths. Math. Phys., Vol. 25, pp. 1-5, (1985).

[10] D. M. Cardoso, *Optimização Linear*, Universidade de Aveiro, (2003).

[11] T. M. Chan, *Optimal Output-Sensitive Convex Hull Algorithms in Two and Three Dimensions*, Departement of Computer Science, University of British Columbia, Vancouver, B.C. V6T IZ4, Canada.

[12] D. R. Chand e S. S. Kapur, *An algorithm for convex polytopes*, JACM 17(1), pp. 778-86, (1970).

[13] J. Chen, *Computational Geometry: Methods and Applications*, Computer Science Departement, Texas A e M University, (1996).

[14] O. L. Chernykh, *Constructing the convex hull of a point set as a system of linear inequalities*, Comput. Maths. Math. Physs., Vol.32, pp. 1085-1096, (1992).

[15] T. H. Cormen, C. E. Leiserson e R. L. Rivest, *Introduction to Algoritms*, McGraw Hill, (1996).

[16] E. D. Demaine, J. S. Mitchell e J. O'Rourke, *The Open Problems Project*.

[17] W. Eddy, *A new convex hull algorithm for planar sets*, ACM Trans. Math. Software 3(4), pp. 398-403, (1977).

[18] Eggleston, *Convexity*, Cambrige University Press, (1958).

[19] L. H. Figueiredo e P. C. P. Carvalho, *Introdução à geometria computacional*, 18º Colóquio Brasileiro de Matemática, Instituto de Matemática Pura e Aplicada, (1991).

[20] L. H. Figueiredo e P. C. P. Carvalho, *Notas de Geometria Computacional*, Instituto de Matemática Pura e Aplicada, (1995).

[21] J. H. Gallier, *Geometric methods and applications: for computer science and engineering*, Springer-Verlag, (2001).

[22] J. H. Gallier, *Convex Sets, Polyhedra and Polytopes: A Deeper Look*, Departament of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104, USA, (2003).

[23] E. Giraldes, V. H. Fernades e M. P. M. Smith, *Curso de Álgebra linear e geometria analítica*, McGraw-Hill, (1995).

[24] J. E. Goodman e J. O'Rourke, *Handbook of Discrete and Computational Geometry*, CRC Press LLC, (1997).

[25] M. Goodrich, L. Guibas e J. Snoeyink, *Computational Geometry Course Notes*.

[26] R. L. Graham, *An efficient algorithm for determining the convex hull of a finit planar set*, Inform. Process. Lett. 1, pp. 132-3,(1972).

[27] P. J. Green e B. W. Silverman, *Constructing the convex hull of a set of points in the plane*, Computer Journal 22, 262-266,(1979).

[28] B. Grunbaum, *Convex Polytopes*, Springer-Verlag, (2003).

[29] R. A. Jarvis, *On the Identification of the Convex Hull of a Finite Set of Points in the plane*, The Australian National University, Departement of Statístics, Box 4, Canberra, A.C.T. 2600, Australia, (1972).

[30] J. L. Kelley, *General Topology*, Springer-Verlag, New York, (1979).

[31] D. G. Kirkpatrick e R. Seidel, *The ultimate planar convex hull algorithm?*, SIAM J. Comput., 15:287-289, (1986).

[32] V. L. Klee, *Extremal Structure of Convex Sets*, Arch. Math., (1957).

[33] D. T. Lee e F. P. Preparata, *The all nearest neighbor problem for convex polygons*, Info. Proc. Lett. 7, pp. 189-192, (1978).

[34] N. L. Martins e A. L. Bajuelos, *Uma classificação de polígonos simples*, Departamento de Matemática, Universidade de Aveiro, (2005).

[35] D. McCallum e D. Avis, *A linear algorithm for finding the convex hull of a simple polygon*, Inform. Process. Lett. 9(1979), pp. 201-206.

[36] A.A. Melkman, *On-line constrution of the convex hull of a simple polyline*, Inform. Proc. Lett. 25(1987), pp. 11-12.

[37] D. Mount, *CMSC 754: Computacional Geometry*, Course Syllabus, Fall 2002, University of Maryland.

[38] J. O'Rourke, *Computational Geometry in C*, Cambridge University Press, Cambridge, 1994.

[39] J. C. de Pina, *Notas de aula Geometria Computacional*, São Paulo, Brasil, (2000).

[40] F. P. Preparata e M. I. Shamos, *Computational Geometry: An Introduction*, New York: Springer-Verlag, (1985).

[41] R. T. Rockafellar, *Convex Analysis*, Princeton, New Jersey, (1970).

[42] V. Santos, *Separação e representação de conjuntos convexos*, Universidade de Aveiro, (1988).

[43] M. I. Shamos, *Computational Geometry*, Ph.D. thesis, Yale University, New Haven, (1978).

[44] A. C. Silva, P. C. Carvalho e M. Gattass, *Diagnóstico de Nódulo Pulmonar Solitário Utilizando Textura e Geometria em Imagens de Tomografia Computadorizada: Resultados Preliminares.*

[45] D. J. Struik, *História Concisa das Matemáticas*, 2ª ed., Lisboa, Gradiva-Publicações, (1990).

[46] G. Toussaint, *A Counter-Example to a Fast Algorithm for Finding the Convex Hull of a Simple Polygon*, Computer Aided Drafting, Design and Manufacturing, Vol. 4, 1-4, (1994).

[47] R. Webster, *Convexity*, Oxford Science Publications, pp. 1-127, (1994).

Índice